تعلم البرمجة مع بيثون ٣

By : Gérard Swinnen

ترجمة : هشام رزق الله

مجتمع لينكس العربي

تعلم البرمجة مع بيثون 3

تأليف : جِرار سوينُ

ترجمة : هشام رزق الله وآخرون

مراجعة و إخراج : مجتمع لينكس العربي

النسخة الإلكترونية من هذه النصوص يمكنك الحصول عليها (باللغة الفرنسيّة) مجانًا وبحرية من:
http://inforef.be/swi/python.htm

بعض فقرات هذا الكتاب قد تم تكييفها وفقا لكتاب:

How to think like a computer scientist ***(كيف تفكر كعالم حاسوب)***

بواسطة: *Jeffrey Elkner، Allen B. Downey* و *Chris Meyers*

متاح على: http://thinkpython.com

أو: http://www.openbookproject.net/thinkCSpy

تمهيد

نحمد الله على منته وفضله في إنجاح هذا العمل، ونرجو أن يكون خالصاً لوجهه الكريم. ونشكر مجتمع لينكس العربي الذي كان المظلة التي عمل تحتها مشروع ترجمة هذا الكتاب، والمنصة التي انطلق منها هذا العمل، والحاضنة التي احتضنت هذا الكتاب. ونتقدم بجزيل الشكر لكل من ساهم معنا في ترجمة أو تنقيح ومراجعة أو تنسيق أو إخراج هذا الكتاب، أو دعمه مادياً أو معنوياً. وإلى كل من كان عوناً لنا بدعمنا معنوياً أو إبداء ملاحظات ساهمت على تحسين سير العمل. ولا ننسى أيضاً أن نتقدم بالشكر لمجتمع البرمجيات الحرّة ومفتوحة المصدر لتوفيرهم الأدوات التي اعتمدنا عليها في تحرير هذا الكتاب. ونأمل أن يكون هذا الكتاب منارة تنير درب السالكين في طريق البرمجة، وبالأخص البرمجة بلغة بيثون، التي تعد لغة هامة للمبرمجين في بيئة نظام التشغيل جنو/لينكس وفي إدارة الخوادم خاصة، وفي أنظمة الحاسوب عامة، وأن يشكل إضافة قيمة إلى المحتوى العربيّ التقني.

لقد اخترنا كتاب **Aprendre à programmer avec Python 3** لما رأيناه من جودة هذا الكتاب، وللمنهجية التي اتبعها المؤلف في ترتيبه، ولغناه بالأمثلة والتمارين العملية على كل موضوع، ولجودة هذه الأمثلة وواقعيتها، ولتسلسله النطقي، وبساطة شرحه. لقد حاولنا قدر استطاعتنا أن نخرج بترجمة ذات جودة عالية قدر المستطاع، ومع هذا يبقى عملاً بشرياً يحتمل النقص، فإذا كانت لديك أية ملاحظات حول حول هذا الكتاب، فلا تتردد بمراسلتنا عبر مجتمع لينكس العربي على http://LinuxAC.org، أو بمراسلة المترجم على بريده <hichemraz[at]gmail.com>.

نأمل أن يساعدك هذا الكتاب عزيزنا القارئ في دراسة وتعلم البرمجة بلغة بيثون، وفي مساعدة الآخرين على تعلم هذه اللغة، وعلى نشر المعرفة بأنواعها. ونأمل كذلك بأن نراك من المساهمين الجادين في صرح البرمجيات الحرة ومفتوحة المصدر، ومن مقدمي البرمجيات والتطبيقات المفيدة عربيًا وعالميًا.

ونرجو لك الاستمتاع بما تتعلمه !

ولا تنسَنا من صالح الدعاء

فريق العمل

الأربعاء، السادس من جمادى الثانية

- 1434 هـ

الموافق 17/4/2013 م

المساهمون :

ترجمة الكتاب : هشام رزق الله

محمد أمين

مراجعة لغوية : أشرف خلف

سيف الإسلام البكري

تصميم الغلاف : أنوار بنشقرون

تنسيق وإخراج : صفا الفليج

عبدالرحيم الفاخوري

إشراف عام : أحمد شريف

تمت ترجمة هذا الكتاب بِدعم من مجتمع لينكس العربي.

مقدّمة المؤلف

بصفتي معلما يدرّس البرمجة بالتوازي مع التخصصات الأخرى، أعتقد أنه يمكنني القول بأن هذا أحد أشكال التعليم غاية في الإفادة والثراء، ووسيلة لتدريب الشباب وبنائهم فكريا، مما له من قيمة عظيمة تتساوى -إن لم تكن تتفوق- مع بعض التخصصات التقليدية مثل دراسة اللاتينية.

إنها فكرة عظيمة إذن، لذا أقترح أن يتم تعلم هذا في بعض القطاعات بما في ذلك التعليم الثانوي. دعونا نكون واضحين: ليس مبكرا تدريب مبرمجين محترفين في المستقبل. ونحن نعتقد ببساطة أن تعلم البرمجة لها مكانتها في التعليم العام عند الشباب (أو على الأقل البعض منهم)، لأنها مدرسة غير عادية من المنطق والصرامة، وحتى الشجاعة.

في الأساس، كُتب هذا الكتاب للطلاب الذين حصلوا على دورة علم "البرمجة واللغات" و"وتكنولوجيا المعلومات" في الصف الثالث للتعليم الثانوي البلجيكي. ويبدو أن هذه الدورة قد تكون مناسبة لأي شخص لم يسبق له أن برمج من قبل، ولكنه يريد تعلم هذا التخصص بنفسه.

نقترح عملية التعلم غير الخطية، وهي بالتأكيد مشكوك فيها. ونحن ندرك أنه سوف تظهر بعض الفوضى في نظر بعض المتمرسين، ولكننا قصدنا ذلك لأننا مقتنعون بأن هنالك العديد من الطرق للتعلم (ليست البرمجة فحسب)، ويجب علينا أن نتقبل على الفور أن الأفراد المختلفين لا يتعاملون مع نفس المفاهيم في نفس الترتيب. ولذلك سعينا قبل كل شيء إثارة الاهتمام وفتح أكبر عدد من الأبواب، ونحن نسعى إلى مراعاة الإرشادات التالية:

- التعليم الذي نسعى إليه عام: نحن نريد تسليط الضوء على ثوابت البرمجة وتقنية المعلومات، دون أن يكون القارئ متخصصا، أو لديه قدرات فكرية غير عادية.
- يجب أن تكون الأدوات المستخدمة عند التعلم حديثة وفعالة، حتى ولو وجب شراؤها، لكن يجب أن تكون قانونية وبأسعار منخفضة للاستخدام الشخصي. وعنواننا في الواقع هو الأولوية للطلاب، وكل خطواتنا للتعلم تهدف إلى إعطائهم فرصة لبدء العمل في أقرب وقت ممكن لتحقيق إنجازاتهم الشخصية التي يمكن أن يطوروها ويستخدموها في أوقات فراغهم.

- وسوف نناقش برمجة واجهة المستخدم الرسومية في وقت مبكر، حتى قبل أن نتعرف على جميع هياكل البيانات المتاحة، لأن هذا النوع من البرمجة يشكل تحديا واضحا في نظر المبرمج المبتدئ. نلاحظ أيضا أن الشباب الآن الذين في صفوفنا الدراسية "منغمسون" بالفعل في الثقافة الحاسوبية المعتمدة على النوافذ وغيرها من الكائنات التفاعلية الرسومية. فإذا اختاروا تعلم البرمجة، فسيكونون حريصين بالضرورة على إنشاء تطبيقات بأنفسهم (ربما بسيطة جدا) حيث نظرة الرسومية موجودة. لقد اخترنا هذا النهج غير العادي قليلا لنسمح للقراء أن يبدؤوا مبكرا جدا في صناعة مشاريعهم الخاصة الصغيرة الجذابة، وليشعروا بقيمة عملهم، ومع ذلك، نطرح جانبا -وعن عمد- بيئات البرمجة المتطورة التي تكتب تلقائيا أسطرًا عديدة من الكود، لأننا لا نخفي التعقيدات الكامنة وراءها.

بعض الأشخاص ينتقدون أن نهجنا لا يركز بما فيه الكفاية على خوارزميات البساطة والوضوح. نحن نعتقد أنه أصبح أقل أهمية مما كان عليه في الماضي. تعلم البرمجة الحديثة يتطلب كائنات تتواصل في أقرب وقت ممكن مع الكائنات والمكتبات للفئات الموجودة. وهكذا يجب أن يتعلم بالسرعة الكافية وفي وقت مبكر التفكير في التفاعلات بين الكائنات، بدلا من بناء الإجراءات، للاستفادة من المفاهيم المتقدمة، مثل الميراث، والتجسيد وتعدد الأشكال.

ولقد قمنا أيضا بتوفير مكان كبير بما فيه الكفاية للتعامل مع أنواع أخرى من هياكل البيانات، لأننا نعتقد أن هذا انعكاس على البيانات يجب عليه أن يظل العمود الفقري لأي تطوير للبرمجيات.

اختيار لغة البرمجة الأولى

هنالك عدد كبير من لغات البرمجة، ولكل منها مزاياه وعيوبه. يجب علينا أن نختار لغة واحدة. عندما بدأنا بالتفكير في هذه المسألة خلال إعدادنا لمنهاج جديد لخيار العلوم والمعلوماتية، تراكمت خبراتنا الشخصية الطويلة في البرمجة بـ Visual Basic (مايكروسوفت) وفي كلاريون (Topspeed). ولقد جربنا أيضا قليلا من دلفي (Borland). ولذلك كان من الطبيعي أننا استخدمنا في البداية لغة واحدة أو أكثر من هذه اللغات. وكان أمام هذه اللغات إذا أردنا استخدامها كأدوات أساسية لتعلم البرمجة العامة اثنان من العوائق الرئيسية:

- ترتبط ببيئات برمجية (معناها برامج) خاصة. وهذا يعني أنه يجب على المدرسة ليس فقط أن تكون على استعداد لشراء ترخيص لاستخدام مثل هذه البرامج لكل محطة عمل (والتي يمكن أن تكون مكلفة)، ولكن حتى بالنسبة للطلاب الذين يرغبون في استخدام المهارات البرمجية في أماكن أخرى خارج المدرسة، وهذا الأمر لا يمكن أن نتقبله. وثمة عيب آخر خطير وهو أن هذه المنتجات تحتوي على "صناديق سوداء" أي أننا لا نستطيع أن نعرف محتواها، ووثائقها ستكون ناقصة وغير مؤكدة.
- هذه اللغات مرتبطة بنظام تشغيل واحد وهو ويندوز. فهي ليست "محمولة" على أنظمة تشغيل أخرى (يونكس، ماك، إلخ). وهذا لا يتناسب مع مشروعنا التعليمي الذي يهدف إلى تعليم عام (وبالتالي متنوع) حيث يتم تسليط الضوء على الثوابت الحاسوبية إلى أقصى حد ممكن.

قررنا بعد ذلك دراسة العرض البديل، وهذا معناه اللغات المقترحة مجانا من قبل حركة البرمجيات الحرة. وجدنا أننا كنا متحمسين، ليس لأنه يوجد في عالم المصادر المفتوحة مفسرات ومترجمات مجانية لمجموعة كبيرة من اللغات، ولكن لأن هذه اللغات حديثة وذات كفاءة عالية ومحمولة (وهذا معناه أنها تستخدم على أنظمة تشغيل مختلفة مثل ويندوز، لينكس وماك)، وهي موثقة توثيقا جيدا.

اللغات السائدة هي بلا شك: سي وسي بلس بلس. هذه اللغة تفرض نفسها بوصفها المرجع المطلق، وكل خبير حاسوب سوف يتعلمها عاجلا أم آجلا. ولكن للأسف هذه اللغة شاقة ومعقدة جدا، وقريبة من الحاسوب. وتركيب جملها ضعيف القابلية للقراءة وقوي الربط. وإن البرامج الكبيرة المكتوبة بلغة سي أو سي بلس بلس طويلة ومرهقة. (وينطبق نفس الشيء على لغة جافا).

من ناحية أخرى، فإن الممارسة الحديثة لهذه اللغة تستخدم على نطاق واسع مولدات التطبيقات وأدوات الدعم المتطورة الأخرى مثل C++Builder و Kdevelop إلخ. ويمكن لهذه البيئات أن تكون فعالة جدا في أيدي المبرمجين ذوي الخبرة، لكنها تقدم العديد من الأدوات المعقدة كثيرا جدا، وهي صعبة على المستخدم المبتدئ والذي من الواضح لا يتقنها. ولذلك سيكون في نظره أنه قد يخفي الآليات الأساسية للغة نفسها. سوف نترك سي وسي بلس بلس لوقت لاحق.

في بداية تعلمنا البرمجة، يبدو من الأفضل أن نستخدم لغة عالية المستوى، وأقل تقييدًا، وتكوين الجمل أكثر قابلية للقراءة. بعد أن فحصنا وواجهنا عدة لغات مثل Perl و Tcl/tk، قررنا أخيرا أن نعتمد على بيثون، لغة حديثة وشعبيتها متزايدة.

تقديم لغة بيثون

> هذه العبارة لـ "ستيفان فرميجيا" المؤرخة منذ زمن قريب، ولكن لا تزال ذات صلة للنص الأساسي. وقد تم نقلها من مقال في مجلة "*Programmez*!" عدد شهر ديسمبر/كانون الأول من سنة 1998. كما أنه متاح على *http://www.linux-center.org/articles/9812/python.html*. و"ستيفان فرميجيا" هومؤسس مشارك لـ *AFUL* (الرابطة الفرنسية لمستخدمي لينكس والبرمجيات الحرة).

لغة بيثون هي لغة محمولة، حيوية (ديناميكية)، مجانية وموسعة، وهي تسمح (ولكنها لا تتطلب ذلك) باتباع نهج الوحدات والبرمجة الشيئية (OOP). تم تطوير لغة بيثون سنة 1989 من قبل غيدو فان روسم وعدد كبير من المتطوعين والمساهمين.

مميزات اللغة

سوف نقوم بوضع المميزات الرئيسية لبيثون مع بعض تفاصيلها:

- لغة بيثون لغة محمولة، وليس فقط على مختلف أنظمة يونكس، ولكن حتى أنظمة تشغيل: ماك، BeOS، NexTStep، MS-DOS ومختلف إصدارات ويندوز. وهنالك مترجم جديد، يدعى JPython، تم كتابته بالجافا ويولد كودبايت جافا.
- بيثون مجانية، ولكن يمكنك استخدامها في المشاريع التجارية دون قيود.

- بيثون مناسبة لسكريبتات من 10 أسطر إلى المشاريع المعقدة التي تحتوي على عشرات الآلاف من الأسطر.
- تكوين جمل بيثون بسيط جدا، ويعمل جنبا إلى جنب مع أنواع البيانات المتقدمة (القوائم والقواميس)، والتي تصنع برامج مدمجة جدا وقابلة للقراءة. وللمقارنة، برنامج بيثون غالبا ما يكون أقصر من 3 إلى 5 مرات من برنامج سي أو سي بلس بلس (أو حتى الجافا) أو ما يعادلها، ووقت تطوير من 5 إلى 10 مرات أقصر وسهل جدا في الصيانة.
- بيثون تدير الموارد بنفسها (الذاكرة، واصفات الملفات) دون تدخل من قِبَل المبرمج عن طريق آلية عد المراجع (مشابهة لجامعي القمامة، لكن مختلفة).
- لا توجد مؤشرات واضحة في بيثون.
- بيثون متعددة الخيوط (اختياري).
- بيثون تدعم البرمجة الشيئية، وهي تدعم الوراثة المتعددة ومشغلات الحمولة الزائدة. في نماذج الكائنات، وعن طريق اتخاذ مصطلحات سي بلس بلس (جميع الأساليب افتراضية).
- بيثون تدعم (مثل الجافا أو الإصدارات الأخيرة من سي بلس بلس) نظام الاستثناءات، الذي يسمح بتبسيط معالجة الأخطاء بشكل كبير.
- بيثون حيوية (ديناميكية) (المفسر يمكنه تقييم السلاسل النصية التي تمثل عبارات أو تعليمات بيثون) ومتعامدة (عدد قليل من المفاهيم كافية لتوليد بنى غنية) وانعكاسية (وهي تدعم الميتابروغراميك، على سبيل المثال، يستطيع الكائن إضافة أو إزالة سمات أو أساليب أو حتى تغيير صنف قيد التنفيذ)، واستقرائية (عدد كبير من أدوات التطوير، مثل المصحح أو المحلل، موجودة في بيثون نفسها).
- مثل Scheme أو SmallTalk، يتم كتابة بيثون بشكل حيوي. جميع الكائنات التي يتم معالجتها من قبل المبرمج يتم تعريف نوع واضح عند التشغيل، والذي لا يحتاج إلى أن تعلن نوعه مسبقا.
- بيثون حاليا هي تطبيقان. الأول، وهو المفسر، حيث سيتم تجميع برامج بيثون في تعليمات محمولة، ثم يتم تشغيلها بواسطة آلة افتراضية (مثل الجافا، مع فارق مهم: يتم كتابة الجافا بشكل ثابت، ويصبح من السهولة تسريع تشغيل برنامج جافا أسرع من بيثون). والثاني يولد مباشرة كود بايت جافا.
- بيثون لغة موسعة: مثل Tcl و Guile، أي أننا يمكننا بسهولة التعامل مع مكتبات سي الموجودة. ويمكننا أيضا أن نستخدمها كلغة موسعة لأنظمة برامج تمديد معقدة.
- إن مكتبات بيثون القياسية، وحزم المساهمة، توفر لك الوصول إلى مجموعة واسعة من الخدمات: سلاسل نصية وتعابير عادية، ومعايير خدمات اليونكس (الملفات، sockets، الخيوط... إلخ)، بروتوكولات الإنترنت (ويب، الأخبار، FTP، CGI، و HTML)، قواعد البيانات وواجهات المستخدم الرسومية.

- بيثون لغة ما تزال تتطور، بدعم من مجتمع المستخدمين والمديرين المتحمسين، ومعظمهم من أنصار البرمجيات الحرة. بالتوازي مع المفسر الرئيسي، المكتوب بلغة سي وهي اللغة التي تم صُنع بيثون بها، ومفسر ثانٍ، مكتوب بالجافا، وهو قيد التطوير.
- وأخيرا، بيثون هي اللغة المختارة لمعالجة الـ XML.

للأستاذ الذي يريد استخدام هذا الكتاب كدعم لدروسه

نحن نأمل مع هذه الملاحظات فتح أكبر عدد ممكن من الأبواب. على مستوى التعليم لدينا، يبدو أنه من المهم إظهار أن برمجة الحاسوب هو عالم واسع من المفاهيم والأساليب، التي يمكن لكل شخص أن يجد مجاله المفضل. نحن لا نعتقد أن جميع الطلاب يجب أن يتعلموا بالضبط نفس الأشياء. نحن نريد أن يكونوا قادرين على تطوير مهاراتهم في مشاريع فردية تختلف إلى حد ما، والتي تسمح لهم بتطوير برامجهم الخاصة وبرامج أقرانهم، وكذلك المساهمة عندما يقترح أحدهم التعاون لعمل كبير.

وعلى أية حال، يجب أن يكون عملنا الرئيسي إثارة الاهتمام، للذي لا يزال بعيدا عن تحصيل حاصل لمادة صعبة مثل برمجة الحاسوب. نحن لا ندعي الاعتقاد بأننا سوف نحمس الشباب على الفور لبناء خوارزميات جيدة. نحن مقتنعون تماما أنه سيتم تثبيت مصلحة عامة بمجرد شعورهم بأنهم أصبحوا قادرين على تطوير مشاريعهم الشخصية، بقدر معين من الاستقلال الذاتي.

ومن هذه الاعتبارات التي أدت بنا إلى تطوير هيكل دراسي والذي يعتقد البعض أن به القليل من الفوضى. سوف نبدأ مع سلسلة من الفصول القصيرة جدا لفترة وجيزة، والتي تفسر ما نشاط البرمجة وتشكل الأساسات القليلة التي لا غنى عنها لتحقيق برامج صغيرة. قد يعتقدون أنه من المبكر البدء بمكتبات الكائنات الرسومية، على سبيل المثال، واجهات المستخدم الرسومية tkinter، بحيث يصبح مفهوم الكائن مألوفا لديهم. وينبغي علينا أن نكون جذابين بما فيه الكفاية للذين يشعرون أنهم اكتسبوا بالفعل إتقان مهارة معينة. ونود حقا أن يتمكن الطلاب من برمجة تطبيق GUI (واجهة المستخدم الرسومية) صغير في نهاية السنة الأولى من الدراسة.

بشكل ملموس جدا، هذا يعني أننا نتوقع استكشاف أول ثمانية فصول من هذه الملاحظات خلال السنة الأولى من الدورة. وهذا يعني أننا سنتناول أولا مجموعة من المفاهيم الهامة (أنواع البيانات والمتغيرات والتعليمات، التحكم في التدفق والدالات والحلقات) بصورة سريعة ودون الحاجة إلى القلق كثيرا على ما يتم فهمه من كل مفهوم قبل الانتقال إلى مفهوم آخر، بدلا من محاولة غرس الذوق الشخصي في البحوث والتجارب. وغالبا ما سيكون أكثر كفاءة لإعادة شرح مفاهيم والآليات المطلوبة في وقت لاحق في حالات وسياقات متنوعة.

في السنة الثانية سوف نسعى إلى تنظيم المعرفة وتعميقها. وسوف يتم تشريح ومناقشة الخوارزميات. وسوف نناقش المشاريع والمواصفات وأساليب التحليل. ونحن نطلب منك دفتر ملاحظات لكتابة تقارير تقنية على وظائف معينة.

والهدف النهائي لكل طالب هو إكمال مشروع برمجي له بعض الأهمية. وسنعمل جاهدين لإنهاء المفاهيم الأساسية النظرية الكافية في وقت مبكر من السنة الدراسية. بحيث يستطيع أي شخص لديه وقت فراغ صنع مشروع.

يجب أن يفهم أن المعلومات المتوفرة في هذه الملاحظات تحتوي على مجموعة واسعة من المجالات (إدارة واجهات المستخدم الرسومية، الاتصالات وقواعد البيانات، إلخ.) اختياريا. وهذه ليست سوى سلسلة من الاقتراحات والمعايير التي أدرجناها لمساعدة الطلاب على اختيار وبدء مشاريعهم الشخصية للتخرج. نحن لا نسعى بأي شكل من الأشكال إلى تدريب متخصصين في لغة معينة أو في مجال تقنية معين: نحن نريد ببساطة إعطاء لمحة عن الفرص الهائلة لأولئك الذين يواجهون المعاناة لتعلم البرمجة بمهارة.

إصدارات اللغة

لغة بيثون ما تزال تُطوّر، لكن الهدف من هذا التطور هو تحسين أو ترقية المنتج. ويجب تعديل البرامج للتكيّف مع النسخ الجديدة التي من شأنها أن تصبح غير متوافقة مع تلك السابقة. والأمثلة في هذا الكتاب تطورت على مدى فترة طويلة نسبيا من الزمن: بعض تم تطويره ببيثون 1.5.2، ثم ببيثون 1.6، 2.0، 2.1، 2.2، 2.3 و 2.4، إلخ. وهم بحاجة إلى تغيير قبل أن يتكيفوا لبيثون 3.

هذا الإصدار الجديد من اللغة، يحمل بعض التغييرات الفنية التي تعطي المزيد من التماسك وسهولة أكبر للاستخدام، ولكن هنالك حاجة إلى تحديث صغير لكافة السكريبتات المكتوبة للإصدارات السابقة. وقد تم إعادة تصميم النسخة الحالية من هذا الكتاب، ليس فقط للتكيف مع أمثلة الإصدار الجديد، ولكن للاستفادة أيضا من هذه التحسينات، والتي هي على الأرجح أفضل وسيلة لتعلم البرمجة اليوم.

إذن قم بتثبيت أحدث إصدار بيثون متاح على نظام التشغيل الخاص بك (بعض الأمثلة لدينا تتطلب الإصدار 3.1 أو أحدث)، واستمتع! ولكن، إذا كنت بحاجة إلى تحليل سكريبتات مُقدَّمة للإصدار السابق، لاحظ وجود أدوات تحويل (انظر خاصة للسكريبت `2to3.py`). والتي موجودة على الإنترنت في موقعنا http://inforef.be/swi/python.htm للإصدار السابق من هذا النص، والتي تم تكييفها للإصدارات السابقة من بيثون، ودائما تستطيع التحميل مجانا.

توزيع بيثون وقائمة المراجع

الإصدارات المختلفة من بيثون (لويندوز ويونكس إلخ)، والدرس التعليمي الأصلي والدليل المرجعي ووثائق مكتبات الدالات إلخ... متوفرة للتحميل مجانا من الإنترنت، من الموقع الرسمي: http://www.python.org

أمثلة الكتاب

يمكنك تحميل الكود المصدري من أمثلة هذا الكتاب من خلال موقع الكاتب:

ن

http://inforef.be/swi/python.htm

أو على العنوان التالي:

http://infos.pythomium.net/download/cours_python.zip

وكذلك على شكل كتاب ورقي مطبوع:

http://www.editions-eyrolles.com

قائمة المحتويات

ص

1

في مدرسة السحرة

تعلُّم البرمجة شيء مهم في حد ذاته يمكن أن يحفز فضولك الفكري، ليس هذا فحسب، بل تعلّمها يفتح أمامك الطريق لإنجاز مشاريع قوية (مفيدة أو مسلّية) والتي ستجعلك في الغالب فخورًا وراضيًا. قبل الدخول في صلب الموضوع سنقترح عليك بعض الملاحظات حول طبيعة البرمجة والسلوك الغريب لمارسيها وشرح بعض مفاهيمها الأساسية. ليس من الصعب تعلم البرمجة لكنها تتطلب منهجًا وقدرًا من المثابرة لأنها علم غير محدود فتواصل التقدم فيه باستمرار.

الصناديق السوداء والتفكير السحري

من الملاحظ في مجتمعنا الحديث أننا نعيش محاطين بالصناديق السوداء بشكل متزايد. من عادة العلماء تحديد أسماء مختلف التقنيات التي نستخدمها بسهولة دون معرفة بنيتها وطريقة عملها بشكل دقيق. مثلا الجميع يعرف كيفية استخدام الهاتف لكن يوجد عدد قليل جدا من التقنيين ذوي درجة عالية من التخصص قادرين على تصميم نموذج جديد.

الصناديق السوداء موجودة في جميع المجالات ومتوفرة للجميع. لا يهمنا هذا عمومًا، بإمكاننا أن نكتفي بمعرفة سطحية لآليتها لاستخدامها دون هواجس. في الحياة اليومية مثلا، لا نهتم فعليًا بمعرفة مكونات البطارية الكهربائية فبمجرد أن نعرف أن البطارية تنتج الكهرباء عن طريق تفاعلات كيميائية ندرك وبسهولة أنها ستنفذ بعد مدة معينة من استخدامها وتصبح بعد ذلك مادة ملوِّثة لا ينبغي رميها في أي مكان، لا حاجة إذن لمعرفة المزيد.

قد يحدث وتصبح بعض الصناديق السوداء معقّدة ولا نستطيع أن نفهمها بقدر كافٍ لنستخدمها بشكل صحيح تحت أي ظرف، قد نميل إلى التمسك بالوقوف ضد ما يرتبط بالتفكير السحري، وهذا يعني شكلا من أشكال الفكر الذي ينطوي على تدخل قوى خارقة للطبيعة أو خصائص لتشرح لعقلنا ما لا يمكن أن نفهمه. هذا ما يحدث عندما يظهر ساحر خفيف اليدين، ونحن نميل إلى الاعتقاد بأن له قوة خاصة، مثل تبرع "مشهد مضاعف"، أو لقبول وجود آليات خارقة ("السائل المغناطيسي"، إلخ)، ونحن لا نفهم استخدامه.

نظرًا لتعقيدها فالحواسيب مثال واضح للصناديق السوداء فحتى لو كنتَ تشعر أنك عشتَ دائما محاطًا بالشاشات ولوحات المفاتيح فغالبًا ليس لديك فكرة كبيرة حول ما يحدث وسط الآلة حقيقة، مثلاً عندما تُحرّك الفأرة سيتحرك على الشاشة -وحسب رغبتك- رسم صغير يشبه السهم، ما الذي يتحرك بالضبط؟ هل تشعر أنك قادر على شرحه بالتفصيل دون نسيان (في جملة أمور) المِجَسّات ومنافذ الواجهات والذاكرات وبوابات المقاييس المنطقية والترانزستورات والبِتات والبايتات وانقطاعات المعالج وشاشات العرض البلوري السائل وشفرة الميكرو والبكسلات وترميز الألوان...؟

في زماننا هذا لا يمكن لأحد أن يدعي أنه متمكن من كافة المعارف التقنية والعلمية المستخدمة لتشغيل الحواسيب. عندما نستخدم هذه الآلات فنحن مجبرون على التعامل معها عقليًا أو على الأقل مع أحد أجزائها، كالأغراض السحرية والتي يحق لنا أن نمارس عليها قوة معينة هي الأخرى سحرية.

مثلاً، نفهم جيّدًا تعليمة كتحريك نافذة تطبيق من خلال شريط العنوان ونعرف تمامًا ما يجب فعله لتنفيذها في العالم الحقيقي، كالتعامل مع الأجهزة التقنية (الفأرة ولوحة اللمس...) التي ستنقل ذبذبات كهربائية من خلال آلية جدّا معقدة ليكون الناتج هو تغيير شفافية أو سطوع جزء من بكسلات الشاشة، لكن في أذهاننا لن تدور أي تساؤلات حول التفاعلات الفيزيائية والدوائر المعقّدة فهذا غرض افتراضي سيتم تفعيله (تحرّك المؤشر على الشاشة) وسيكون بمثابة عصا سحرية للتحكم في الغرض الذي بدوره افتراضي وسحري أيضًا (نافذة التطبيق). التفسير العقلاني لما يحدث فعلاً داخل الجهاز تراجع إذن لصالح استنتاج متصوّر أكثر سهولة لكنه في الحقيقة مجرّد وهم.

إن كنتَ مهتمًا بالبرمجة فاعلم أنك ستواجه باستمرار أشكال مختلفة من هذا "التفكير السحري" ليس مع الأشخاص الآخرين فحسب (كمَن يَطلب منك إنشاء برنامج معين) بل وأيضًا مع تصوّراتك الذهنية خاصة. يجب عليك أن تزيل هذه الأوهام الزائفة والتي هي في الحقيقة مجرد تخمينات والاعتماد على تفسيرات مجازية مبسطة من الواقع حتى تستطيع تسليط الضوء ولو جزئيًا على الأثار العلمية الحقيقية.

وهذا متناقض إلى حد ما ويفسر عنوان هذا الفصل، أي مع تطور مهارتك ستسيطر على الجهاز أكثر وبالتالي ستصبح مع مرور الوقت كساحر في أعين الناس!

أهلا بك إذن في مدرسة السحرة مثل الشهير هاري بوتر!

السحر الأبيض والسحر الأسود

ليست لدينا النيّة بالطبع لمساواة البرمجة بعلم التنجيم وإن كنا نرحب بك هنا كساحر مبتدِئ فهذا فقط لإثارة انتباهك على الأثار المترتبة على هذه الصورة التي قد تعطيها لنفسك (ربما عن غير قصد) لمعاصريك. قد يكون من المثير للاهتمام أن تستعير بعض الكلمات من المفردات السحرية لإظهار التعجب من هذه الممارسات.

البرمجة هي فن تلقين جهاز لينجز مهاما جديدة لم يكن قادرًا على تنفيذها سابقًا، تمنحك مزيدًا من سيطرة على كافة الأجهزة المرتبطة بالشبكة وليس على جهازك فحسب. بطريقة أخرى يمكن مساواة هذا بشكل خاص من السحر يعطي القوة لمن يمارسه، وهي غامضة بالنسبة للكثيرين وحتى مثيرة للقلق عندما ندرك إمكانية استخدامها لأغراض غير شريفة.

في عالم البرمجة نقصد بمصطلح هاكر المبرمجين المحنّكين الذين قاموا بتحسين أنظمة التشغيل الشبيهة بيونكس ووضع تقنيات الاتصال التي كانت أساس التطور المذهل للإنترنت وهم مستمرون أيضًا في إنتاج وتحسين البرمجيات الحرّة/المصادر المفتوحة، إذن حسب ما سردناه؛ فالهاكرز هم أسياد السحر يمارسون السحر الأبيض.

ولكن هنالك مجموعة أخرى من الناس أطلقت عليهم الصحافة عن طريق الخطأ على أنهم الهاكرز، حين كان يجب عليهم أن يسمونهم كراكرز. وهؤلاء الناس يطلقون على أنفسم اسم الهاكرز لأنهم يريدون أن نعتقد على أنهم محترفون للغاية، ولكن بشكل عام إنهم بالكاد متخصصون، ولكنهم ضارون للغاية، لأنهم يستخدمون معرفتهم للعثور على بعض ثغرات في أنظمة الحاسوب (التي تم صنعها من الهاكرز الحقيقيين) لتنفذ جميع عملياتهم الغير قانونية مثل: سرقة معلومات سرية والاحتيال، إرسال الرسائل الغير مرغوب بها (سبام) والفيروسات والمواد الإباحية والتقليد وتدمير المواقع ...إلخ. طبعا هؤلاء السحرة منحرفين بشكل خطير في السحر الأسود. وهناك أيضا مجموعة أخرى، القراصنة الحقيقيون يسعون إلى تعزيز أخلاقهم التي تقوم أساسا على المحاكاة وتبادل المعلومات (معرفة) والذين يستطيعون بناء برامج تتسم بالكفاءة وهي أيضا بالتأكيد تكون أنيقة وذات بنية نظامية تماما ولديها وثائقها، ستكتشف بسرعة أنه من السهل إنتاج الكثير من البرامج التي تعمل ولكن أغلبها غامضة ومربكة وغير مفهومة لغير مبرمجيها. هذا النوع من البرمجة يكون في الغالب غامضًا ويطلق عليها من قبل الهاكرز بالسحر الأسود.

نهج المبرمج

مثل الساحر، يكون للمبرمج قوة غريبة، على سبيل المثال تستطيع تحويل جهاز إلى جهاز آخر وآلة حاسبة إلى آلة كاتبة أو رسم وقليل من السحر بعد تستطيع تحويل الأمير إلى ضفدع وذلك باستخدام لوحة المفاتيح لتدخل بعض التعويذات الغامضة مثل الساحر. وهو قادر على علاج التطبيقات سيئة أو تلقي فترات عن طريق الانترنت. لكن كيف يكون هذا ممكنا؟

وقد يبدو هذا متناقضا، كما لاحظنا سابقا، المعلم الحقيقي هو في الواقع الذي لا يؤمن بأي سحر وأي هبة ولا تدخل خارق. فيظل باردًا وعنيدًا يسعى وراء المنطق غير المريح.

تفكير المبرمج يجمع بين البنى الفكرية المعقدة، المماثلة لتلك التي قام بها علماء الرياضيات والمهندسون والعلماء. كما في الرياضيات فإن البرمجة تستخدم لغات رسمية لوصف المنطق (أو الخوارزميات) مثل المهندس، الذي يصنع الأجهزة، فيجمع عناصر لتنفيذ آليات وتقييم أداءها. مثل العالم يلاحظ سلوك النظام المعقد ويصنع النماذج ويختبرها.

النشاط الأساسي للمبرمج هو حل المشاكل.

ويجب أن يكون على مستوى عالٍ من الكفاءة، التي تشمل مختلف المهارات والمعارف، وأن يكون قادرا على إعادة صياغة المشكلة بطرق عديدة ومختلفة ويجب أن يكون قادرا أيضا على وضع حلول مبتكرة وفعالة، وقادرا على التعبير عن هذه الحلول بوضوع وكمال. كما ذكرنا سابقا، يجب علينا أن نلقي الضوء على الآثار العملية العقلية "سحرية" البسيطة أو الملخصة جدا.

البرمجة هي في الواقع "شرح" بالتفصيل للجهاز ما يجب عليه القيام به، مع العلم أن الجهاز لا يفهم لغة الإنسان، ولكن فقط تنفيذ معالجة آلية لتسلسل من الأحرف، في الغالب يتم التعبير عن رغبة في تحويل المصدر من حيث "السحرية" إلى منطق صحيح منظم تماما وكل تفاصيله مفهومة، وهذا ما يسمى بالخوارزمية.

على سبيل المثال أنظر إلى هذه السلسلة من الأرقام بالترتيب: 12،5،36،21،15،23،19،47 ... كيف لنا أن نحصل من الحاسوب على ترتيب هذه الأرقام؟

الرغبة "السحرية" ليست فقط النقر على الزر، أو إدخال تعليمة واحدة في لوحة المفاتيح ليتم ترتيب كل رقم في مكانه. بل إن عمل المبرمج هو صنع البرنامج لترتيبه ويجب عليه شرح جميع الخطوات لعملية الفرز (في الواقع هنالك طريقة فريدة أو أكثر لعمل هذا) ويجب علينا ترجمة جميع الخطوات في سلسلة من التعليمات البسيطة، مثلا "قارن أول رقمين، وبدلهما إذا لم يكونا في الترتيب المطلوب، ثم ابدأ مع غيره، (الثاني والثالث...إلخ)".

إذا كانت التعليمات، وذلك حتى ضوء بسيطة بما فيه الكفاية، يمكن ترميزها في الجهاز وفقا لمجموعة صارمة للغاية من الاتفاقيات المقررة سابقا، والمعروفة باسم لغة الحاسوب، لفهم هذا يجب توفير جهاز مع وجود برنامج لترجمة هذه التعليمات من خلال ربط كل كلمة مع عمل معين للغة. وهكذا فقط يمكن أن يحقق السحر.

لغة الآلة، لغة البرمجة

بالمعنى الحرفي للكلمة، الحاسوب ما هو إلا جهاز ينفذ عمليات بسيطة عن طريق إشارات كهربائية متسلسلة، يتم شحنها عن طريق إشارتين لا غير (على سبيل المثال أقصى أو أدنى جهد محتمل) هذه الإشارات المتتالية يستطيع فهمها الحاسوب بالمنطق "كل شيء أو لا شيء" ويمكن أن تعتبر تقليدية حسب تسلسل الأرقام بدئا من القيمتين 0 و 1. يطلق عليها بالنظام الرقمي وهي تقتصر على رقمين اثنين، وتسمى أيضا النظام الثنائي.

لقد عرفنا الآن أن الحاسوب في عملياته الداخلية يتعامل مع الأرقام الثنائية، وهو لا يتعامل مع أي شيء آخر. لذا يجب تحويل جميع المعلومات (المشفرة أيضا) إلى الشكل الثنائي. وهذا لا ينطبق فقط على البيانات التي نود معالجتها (النصوص والصور والأصوات والأعداد) بل وينطبق حتى على البرامج (التعليمات المتتالية التي ستخبر الحاسوب ماذا يفعل).

واللغة الوحيدة التي يفهمها الحاسوب بالفعل بعيدة جدا عن اللغة التي نستخدمها، حيث إنها سلسلة طويلة من 1 و 0 (تسمى بتات) ويتم التعامل معها على شكل مجموعات: ثمانية (بايت) أو 16 أو 32 أو حتى 64. و"لغة الآلة" هذه غير مفهومة غالبا بالنسبة لنا. وللتخاطب مع جهاز الحاسوب، فسوف نستخدم أنظمة ترجمة آلية قادرة على تحويل الكلمات التي نفهمها (عادة التي باللغة الإنجليزية) إلى سلاسل من الأعداد الثنائية، لتكون ذات معنى بالنسبة لنا. ويتم إنشاء نظم الترجمة هذه على أساس مجموعة من المصطلحات، وسيكون واضحا العديد من الاختلافات. يطلق على نظم الترجمة اسم المترجم أو المفسر حسب الطريقة التي استخدمتها لتنفيذ الترجمة.

إننا ندعو مجموعة من كلمات مفتاحية (مختارة تعسفيا) من لغة البرمجة مرتبطة مع مجموعة من القواعد المحددة حول كيفية تجميع هذه الكلمات لتصبح عبارات يمكن للمفسر أو المترجم ترجمتها إلى لغة الآلة (الثنائية). لكل لغة برمجة مستوى معين فمثلا اللغات ذات المستوى المنخفض (على سبيل المثال لغة التجميع) أو ذات المستوى العالي (على سبيل المثال باسكال وبيرل وبيثون...) اللغات ذات المستوى المنخفض تتكون من الإرشادات الأساسية جدا جدا "القريبة من لغة الآلة" واللغات رفيعة المستوى تكون أكثر تجريدا وأكثر "قوة" وبالتالي أكثر "سحرًا". وهذا يعني أنه يمكن أن تترجم كل هذه التعليمات من قِبل المترجم أو المفسر بعدد كبير من تعليمات الجهاز البدائية. لغة البرمجة التي ستتعلمها أولا هي بيثون، بيثون هي لغة برمجة عالية المستوى، والتي تترجم إلى رمز ثنائي معقد ودائما ما يأخذ بعض الوقت. قد يبدو أن هذا الوضع يُضعف اللغة، لكن الحقيقة أن اللغات ذات المستوى العالي تتميز بمميزات هائلة: إن كتابة برنامج بلغة رفيعة المستوى أسهل كثيرا ويستغرق وقتا أقل لكتابته واحتمال الأخطاء فيه قليلة، وصيانته (وذلك معناه المساهمة في التعديلات اللاحقة) والبحث عن الأخطاء "العلل" سهلٌ للغاية. بالإضافة إلى أن البرنامج المكتوب بلغة عالية المستوى يكون غالبا "محمول" هذا يعني أن نغير تغيرات قليلة في البرنامج ليعمل على عدة أجهزة وأنظمة تشغيل مختلفة. والبرنامج المكتوب بلغة ذات مستوى منخفض لا يمكن أن يعمل إلا على نوع واحد فقط من الأجهزة، ويجب أن يتم إعادة كتابته كاملا ليعمل على جهاز آخر. ما شرحناه في فقرة مختصر: ربما لاحظتم العديد من "الصناديق السوداء": مترجم ونظام تشغيل واللغة وتعليمات الجهاز والكود الثنائي وما إلى ذلك.

إن تعلم البرمجة سوف يسمح لك بفتح جزئي، ومع ذلك لا يمكنك أن تقوم بفتح كلّي. العديد من كائنات المعلوماتية التي تم صنعها بواسطة أشخاص آخرين يمكن بالنسبة لك أن تظل "سحرية" لفترة طويلة (بدءا من لغة البرمجة نفسها، على سبيل المثال). ويجب عليك أن تثق بأصحابها، وربما تصاب أحيانا بخيبة أمل من أن النتيجة ليست دائما ما تستحق الثقة. إذن كن يقظا، وتعلم كيفية التحقق من الوثائق. في مشاريعك الخاصة، كن حذرا وتجنب بأي ثمن استعمال "السحر الأسود" (البرامج الممتلئة بالحيل التي تفهمها أنت فقط): فإن الهاكر الجدير بالثقة ليس لديه ما يخفيه.

تعديل المصدر - المفسر

البرنامج الذي نكتبه في أي لغة برمجة هو نص بسيط، يمكنك البحث عن كل أنواع البرامج الأكثر والأقل تعقيدا وسوف تجد أنها تنتج فقط نصا بسيطا وهذا يعني دون تنسيق أو سمة نمط معين (يعني لا مواصفات للخط، ولا عناوين ولا خط غامق أو مسطر أو مائل ...إلخ)[1]. النص الذي تنتجه يسمى الآن بالشيفرة المصدرية. كما ذكرنا سابقا، لابد من ترجمة تعليمات البرنامج المصدر إلى سلسلة من التعليمات الثنائية مفهومة مباشرة من قبل الجهاز: "شيفرة الكائن في حالة بيثون" ويتم دعم هذه الترجمة بواسطة مفسر تم ترجمته سابقا. هذا الأسلوب الهجين (الذي يستخدم أيضا من قبل لغة جافا) يهدف إلى تعظيم فوائد التفسير والتجميع مع تقليل عيوب كل منهما. يرجى منك أن تبحث عن كتاب يشرح هاتين التقنيتين في حالة تريد أن تعرف المزيد. اعلم أنه يمكنك صنع برامج عالية الأداء مع بيثون، على الرغم من أنه لا جدال في أن اللغة المترجمة بدقة مثل لغة سي يمكنها أن تفعل أفضل من حيث توقيتها.

وضع البرنامج - البحث عن أخطاء (تصحيحها)

البرمجة عملية معقدة جدا، كما هو الحال في أي نشاط بشري، في بعض الأحيان قد نقع في أخطاء كثيرة لأسباب عديدة وهي تسمى أخطاء البرمجة "العلل"[2] وجميع التقنيات التي يتم تنفيذها لكشفها وتصحيحها تسمى بالتصحيح. في الحقيقة؛ قد يكون في البرنامج ثلاثة أنواع مختلفة من الأخطاء وينبغي أن نتعرف عليها لنميزها.

أخطاء التعليمات

يمكن تشغيل برنامج بيثون في حالة كان خاليا من الأخطاء، خلافا لذلك فإن عملية الترجمة ستتوقف وتحصل على رسالة خطأ. "بناء الجملة" هي مصطلح يشير إلى قواعد اللغة التي وضعها مبرمجوها ولقد وضعت لهيكلة البرنامج. كل لغة لديها بناء لغة فمثلا في الفرنسية على سبيل المثال يجب على الجملة أن تبدأ بحرف كبير وتنتهي بنقطة. وهذه الجمل لديها خطئان في بناء الجملة. في النص العادي، وجود بعض الأخطاء في بناء الجملة هنا وهنالك لا يهم. قد يحدث هذا (في الشعر مثلا) كما ترتكب الأخطاء النحوية عن طريق الخطأ وهذا لا يعني أننا لا نستطيع فهم النص. في المقابل في جهاز الحاسوب، أي خطأ لغوي يقع، فإنه يحدث دائما تحطم، بالإضافة إلى عرض رسالة خطأ، خلال الأسابيع الأولى من مسيرتك للتعلم سوف تجد بالتأكيد العديد من الأخطاء وسوف تبقى وقت طويلا لتصحيحها. ولكن المحترفين سيفعلون ذلك في وقت أقل من ذلك بكثير.

---

1 وتسمى هذه البرامج برامج معالجة النصوص. على الرغم من أنها توفر مجموعة متنوعة من الأتمتة، وغالبا ما تكون قادرة على إبراز بعض عناصر النص المعالج (تلوين، تركيب الجملة، على سبيل المثال)، لا تحدث بدقة سوى على النص غير المنسق. فهي مختلفة تماما عن برامج معالجة النصوص، التي تتمثل مهمته على وجه التحديد تخطيط وتجميل النص مع سمات من أي نوع، بطريقة تجعله قابل للقراءة قدر الإمكان.

2 bug هي مصطلح إنجليزي الأصل يُستخدم لوصف الحشرات الصغيرة المزعجة مثل البق. الحواسيب الأولى التي تعمل بمساعدة المصابيح والراديو التي تتطلب كهرباء ذات فولتية عالية نسبيا، تم حرقها عدة مرات بسبب دخول هذه الحشرات الصغيرة إلى الدوائر المعقدة فتسبب أجسادها دوائر قصيرة ثم عطب غير مفهوم.

ضع في اعتبارك أن الكلمات والرموز ليس لها معنى في حد ذاتها بل هي مجرد تسلسل من الرموز التي يجري تحويلها تلقائيا إلى أرقام ثنائية. لذلك يجب أن نكون حذرين للغاية للحفاظ على النحو الصارم في بناء الجملة اللغوية. أخيرا، تذكر أن كل التفاصيل مهمة (على سبيل المثال: الحرف الكبير والحرف الصغير) وعلامات الترقيم ويمكن لأي خطأ -مهما كان صغيرا مثل نسيان فاصلة- أن يغير الكثير من معنى الجملة ويسبب لك مشاكل كثيرة. فمن حسن حظك أن تتعلم لأول مرة مع بيثون التفسيرية. لأن البحث عن الأخطاء سيكون أمرا سهلا للغاية. ولكن مع اللغات التي تستعمل مترجما مثل سي بلس بلس يجب عليك إعادة ترجمة البرنامج بأكمله بعد كل تغيير مهما كان صغيرا.

دلالات الأخطاء

أما النوع الثاني من الأخطاء هو الخطأ المنطقي أو الخطأ الدلالي. إذا كان هناك هذا النوع من الأخطاء في أحد البرامج فيتم تشغيل هذا البرنامج بدون أن تحصل على أية رسالة خطأ ولكن النتيجة غير متوقعة، فلقد كنت تقصد شيئا آخر. في الواقع، ينفذ البرنامج ما طلب منه بالضبط، لكن المشكلة أنه لا يطابق ما تريد أن تفعله أنت. تسلسل التعليمات في البرنامج لا يلبي الهدف. المنطق أو الدلالة خاطئة. إن البحث عن الأخطاء المنطقية مهمة شاقة جدا. من شأنها أن تثبت قدرتك على إزالة أي شكل من "التفكير السحري" في حججك. وسوف تحتاج إلى الصبر ويجب عليك كتابة سطر وراء سطر في المفسر لتعرف أين الخطأ المنطقي.

أخطاء وقت التشغيل

النوع الثالث من الأخطاء هو خطأ وقت التشغيل والذي يظهر فقط عندما يكون البرنامج قيد التشغيل، ولكن يجب أن تنشأ ظروف خاصة ليظهر هذا الخطأ (على سبيل المثال، عندما يحاول البرنامج قراءة ملف لم يعد موجودا)، وتسمى هذه الأخطاء بالاستثناءات لأنها تشير إلى شيء استثنائي (ولكنه وقع) وسوف تواجه هذه الأخطاء عند جدولة المشاريع الكبيرة وسوف نتعلم في وقت لاحق من الدورة كيفية التعامل مع هذه الأخطاء.

البحث عن الأخطاء والتجارب

من أهم المهارات للحصول على تدريب خاص بك هو وضع البرنامج في حالة "debuging" (تصحيح). هذا النشاط قد يصيبك بالجنون أحيانا لكنه دائما غني بالمعلومات، حيث ستتعلم الكثير من الأشياء والأفكار. وهذا العمل هو مماثل في كثير من النواحي لتحقيق الشرطة. حيث يمكنك تفحص مجموعة من الحقائق ويجب عليك أن تضع فرضيات عمليات التفسير وإعادة بناء الأحداث التي أدت منطقيا إلى حدوث هذه النتائج التي نراها.

ويرتبط هذا النشاط أيضا بالعمل التجريبي في العلوم. حيث يمكنك الحصول أولا على ماهية الخطأ، ويمكنك تغيير برنامجك والمحاولة مرة أخرى وجعله فرضية تتيح لك التنبؤ بماهية التغييرات التي ستغيرها. إذا كان التنبؤ صحيحا فستتقدم خطوة

إلى الأمام نحو برنامج يعمل بدون مشاكل وإذا كان التنبؤ يثبت الخطأ يجب عليك إجراء فرضية جديدة. كما قال شرلوك هولمز: "عندما تقضي على المستحيل، ما يتبقى، حتى لو كان هذا غير محتمل، يجب أن يكون الحقيقة" (أ. كونان دويل، البرج رابع).

بالنسبة لبعض الناس، "البرمجة" و"التصحيح" يبدوان لهم نفس الشيء، وهذا معناه أن نشاط البرمجة هو في الواقع تعديل وتصحيح البرنامج نفسه باستمرار، حتى يعمل كما تريده. والفكرة هي بناء برنامج يبدأ دائما مع الخطوط العريضة التي هي بالفعل شيئا ما (والتي يتم تصحيحها بالفعل)، والتي يتم إضافة طبقة طبقة من التغييرات الصغيرة، وتعديل الأخطاء تدريجيا وذلك يتم في كل مراحل من عملية البرنامج الذي يعمل.

على سبيل المثال أنتم تعلمون أن لينكس هو نظام تشغيل (يعني برنامج كبير) وهو يحتوي على الآلاف من الأسطر من التعليمات البرمجية، ولقد بدأ لينوس تورفالدس ببرنامج صغير بسيط لاختبار ملامح من إنتل 80386. ووفقا للاري غرينفيلد "دليل مستخدم لينكس" إصدار بيتا: "وكان من بين المشاريع الأولى للينوس هو برنامج لتحويل سلسلة AAA إلى BBB وهذا ما أصبح في نهاية المطاف نظام تشغيل لينكس". ولا تعتقد أننا نريد أن ندفعك لتعلم البرمجة عن طريق التجربة والخطأ للتقريب من فكرة غامضة عندما تبدأ مشروع برمجي له حجم معين، يجب أن تقوم بعمل مواصفات بأدق تفصيل ممكن، وعلى أساس متين يبنى عليه البرنامج. توجد أساليب مختلفة للتحليل. لكن هذه الدراسة هي خارج هذه الملاحظات، ومع ذلك سنقدم في وقت لاحق بعض الأفكار الأساسية (أنظر للفصل 15).

2

الخطوات الأولى

البرمجة هي فن قيادة الحاسوب ليفعل ما تريده بالضبط، وبيثون هي واحدة من اللغات القادرة على فهمك من أجل الحصول على أوامرك، سنحاول أن نبدأ فوراً بأوامر بسيطة جدا ألا وهي الأرقام؛ لأن الأرقام هي المفضلة لدي، وسنقدم أول "التعليمات"، ونحدد طريقة تعريف بعض مفردات الحاسوب الأساسية، التي سنعمل عليها .

وكما شرحنا في المقدمة (أنظر إصدارات اللغة، الصفحة الثامنة)، لقد اخترنا استخدام الإصدار الجديد من بيثون، ألا وهو الثالث، والذي أُدخلت عليه بعض التغييرات النحوية خلافاً للإصدارات السابقة. وسأخبركم -متى كان ممكناً- بالاختلافات بين إصدارات بيثون حتى تتمكنوا من تحليل أو استخدام برامج قديمة مكتوبة ببيثون 1 أو 2.

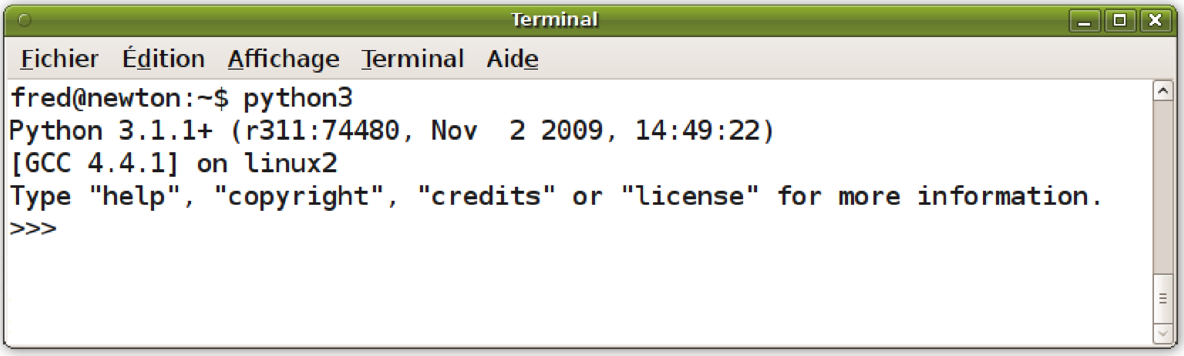
الحساب مع بيثون

تتميز بيثون بأنها تُستخدم بطرق عدة، وسنستخدم بيثون أولا في وضع تبادلي وهذا يعني التحدث مع بيثون مباشرة عبر لوحة المفاتيح. وهذا سيجعلنا نكتشف سريعا مميزات لغة بيثون. ثم سنتعلم كيفية إنشاء برامجنا الأولى النصية (سكريبتات) وحفظها على القرص.

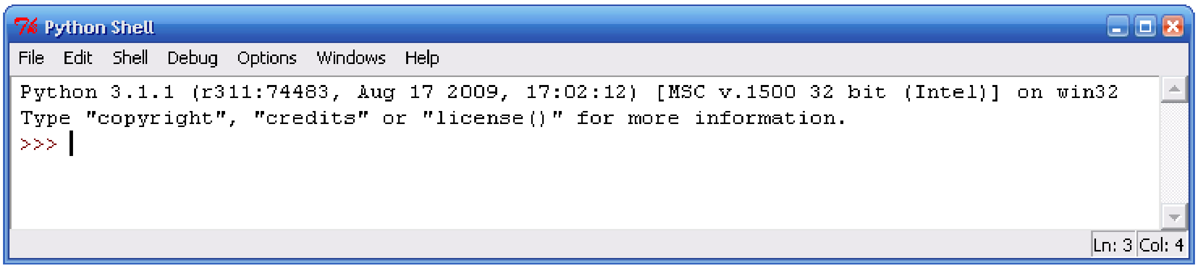
يمكن تشغيل المترجم من الطرفية مباشرة (في لينكس "shell" أو في نافذة "dos" في نظام التشغيل ويندوز) نحتاج إلى كتابة الأمر **python3** فقط (نفترض أن البرنامج كان مثبتاً تثبيتاً صحيحاً وإصدار بيثون هو 3) أو نكتب **python** فقط (إذا كان إصدار بيثون المثبت على جهازك أقدم من الإصدار 3)

إذا كنت تستخدم واجهة رسومية مثل ويندوز، جنوم، كدي أو WindowMaker فربما تفضل العمل في "نافذة الطرفية" أو في محطة عمل مثل IDLE على سبيل المثال، هذه صورة لنافذة الطرفية لواجهة جنوم في نظام تشغيل لينكس (توزيعة أبونتو)[3] :

3 في نظام تشغيل ويندوز، في الغالب يجب عليك الاختيار بين بيئة IDLE التي تم تطويرها من قبل غويدو فان روسيم، والتي نفضلنا نحن، وبين PythonWin، واجهة تطوير تم تطويرها من قبل مارك هامونك. وتوجد أيضا بيئات أخرى أكثر تطورا، مثل Boa Constructor على سبيل المثال (التي تعمل بشكل مماثل لدلفي)، لكننا نعتقد أنه غير مناسب جدا للمبتدئين. لمزيد من



في IDLE في ويندوز، سيكون مكان عملك كهذا:



الإشارة < "أكبر من" المكررة ثلاثا هي إشارة سريعة أو موجِّه فوري يخبرك أن بيثون مستعد لتلقي الأوامر.

على سبيل المثال، تستطيع استعمال المفسر على أنه آلة حاسبة بسيطة لسطح المكتب. أرجو منك أن تختبر الأوامر التالية بنفسك (تعود على استعمال كراس التمارين لكتابة النتائج التي تظهر على الشاشة):

```
>>> 5+3
>>> 2 - 9          # المسافات اختيارية
>>> 7 + 3 * 4      # التسلسل الهرمي للعمليات الحسابية
>>> (7+3)*4
>>> 20 / 3         # انتباه: هذا يعمل بشكل مختلف في بيثون 2
>>> 20 // 3
```

كما ترون، فإن العوامل الحسابية للجمع والطرح والضرب والقسمة هي على التوالي + و- و* و/. وما بين القوسين نتيجته متوقعة:

المعلومات يرجى زيارة موقع بيثون.

في نظام تشغيل لينكس، نحن نفضل شخصيا العمل في نافذة الطرفية البسيطة لتشغيل مفسر بيثون أو تشغيل السكريبتات، واستخدام محرر نص عادي مثل Gedit أو Kate أو واحد أكثر تخصصا مثل Geany.

> في بيثون 3، عامل القسمة / يقوم بقسمة حقيقية (عدد حقيقي). وإذا أردت أن تحصل على قسمة عدد صحيح، يجب عليك استخدام المعامل //. ويرجى ملاحظة أن هذه التغييرات أدخلت على تكوين جملة بيثون 3- مقارنة مع الإصدارات السابقة. فإذا كنت تستخدم إحدى هذه الإصدارات، يرجى ملاحظة أن العامل / يقوم بقسمة عدد صحيح بشكل إفتراضي، إذا قمت بتمرير برامترات أعداد صحيحة وقسمة عدد حقيقي، إذا أدخلت (على الأقل) عددا حقيقيا واحدا. لحسن الحظ تم التخلي عن السلوك القديم من بيثون، لأنها كانت تؤدي إلى خلل في بعض الأحيان من الصعب اكتشافه.

```
>>> 20.5 / 3
>>> 8,7 / 5            # خطأ!
```

يرجى ملاحظة أن هنالك قاعدة تُطبَّق في جميع لغات البرمجة هي المصطلحات الرياضيات السارية في البلدان الناطقة باللغة الإنجليزية ومن بينها الفاصلة العشرية التي هي دائما نقطة، وليست فاصلة كما عندنا. وسوف نلاحظ أيضا أن في عالم الحاسوب يشار غالبا إلى الأعداد الحقيقية كأرقام "النقطة العائمة".

البيانات والمتغيرات

سيكون لدينا الفرصة لتعلم تفاصيل أكثر لأنواع مختلفة من البيانات الرقمية. ولكن قبل ذلك سنتناول الآن مفهوما أكثر أهمية. العمل الرئيسي الذي يقوم به برنامج الحاسوب هو معالجة البيانات. ويمكن لهذه البيانات أن تكون متنوعة جدًا (في الواقع، كلها رقمية[4])، لكن في ذاكرة الحاسوب تحول دائما في النهاية إلى تسلسل محدد من الأرقام الثنائية. للوصول إلى البيانات يقوم برنامج الحاسوب (بغض النظر عن اللغة المستخدمة في كتابته) باستخدام أعداد كبيرة من المتغيرات مختلفة الأنواع. المتغير يظهر في لغة البرمجة كأي اسم متغير آخر تقريبا (أنظر أدناه)، ولكن عند الحاسوب يكون مرجعا يشير إلى عنوان ذاكرة، وهذا معناه موقع محدد في ذاكرة الوصول العشوائي (ram). في هذا الموقع يتم تخزين قيمة محددة (وهي البيانات) في سلسلة من الأرقام الثنائية، ولكن ليس بالضرورة رقم واحد في نظر لغة البرمجة المستخدمة، ويمكن أن يتم هذا عن طريق أي "كائن" ويتم وضعه في ذاكرة الحاسوب. على سبيل المثال: عدد صحيح، عدد حقيقي، سلسلة نصية، ناقل، سلسلة مطبعية، جدول أو وظيفة ... إلخ، سوف نشرح في الصفحات التالية التمييز بين كل هذه المحتويات الممكنة ولغة البرمجة المستخدمة لأنواع مختلفة من المتغيرات (عدد صحيح، عدد حقيقي، سلسلة نصية، قائمة ... إلخ).

4ماذا يمكن فحصه بالضبط؟ هذا السؤال مهم جدا، ويجب عليك البحث في دراستك في العلوم العامة.

أسماء المتغيرات والكلمات المحجوزة

أسماء المتغيرات هي الأسماء التي تستطيع تسميتها بحرية (تقريبا)، حاول اختيارهم بشكل جيد ويفضل أن يكون قصيرا جدا وواضحا بقدر الإمكان؛ وذلك للتعبير بوضوح عما يفترض أن يحتوي المتغير، على سبيل المثال، أسماء المتغيرات مثل الارتفاع أو الارتفاع البديل أكثر ملائمة للتعبير من ارتفاع x.

المبرمج الجيد هو الذي يجعل أسطر برنامجه سهلة القراءة.

في بيثون يجب أن تتبع أسماء المتغيرات بعض القواعد البسيطة:

- اسم المتغير يتكون من سلسلة من الحروف (A إلى Z الكبيرة) و(a إلى z الصغيرة) والأرقام (0 إلى 9) ويجب أن يبدأ دائما بحرف.
- لا يسمح إلا بالأحرف العادية ويمنع استعمال المساحات والرموز (أحرف خاصة مثل!@#$%^&* وما إلى ذلك) باستثناء الرمز _ (خط تحت السطر).
- من المهم التمييز بين الأحرف الكبيرة والصغيرة.

انتبه: JOSEPH،joseph،Joseph متغيرات مختلفة فكن حذرا .

تعود على كتابة معظم الأسماء بأحرف صغيرة. هذه مجرد اتفاقية ولكنها باحترام واسع. استخدم الأحرف الكبيرة ضمن نفس الاسم، ذلك لزيادة سهولة القراءة، كما هو الحال في TableofContents.

بالإضافة إلى تلك القواعد، يجب أن نضيف أيضا أنه لا يمكننا استخدام أسماء متغيرات محجوزة (33 كلمة) المبينة أدناه؛ يتم استخدامها من قبل اللغة نفسها:

and	as	assert	break	class	continue	def
del	elif	else	except	False	finally	for
from	global	if	import	in	is	lambda
None	nonlocal	not	or	pass	raise	return
True	try	while	with	yield		

تعيين قيمة متغير

لقد أصبحنا نعرف الآن كيف نختار اسم المتغير بحكمة، الآن سوف نشرح كيفية تعيين قيمة للمتغير والمصطلح "تعيين قيمة" هي دلالة على عملية إقامة صلة بين اسم المتغير وقيمته (محتوياته). في بيثون، كما هو الحال في العديد من اللغات الأخرى، تمثل هذه العملية بواسطة علامة المساواة (=):

```
>>> n = 7                       # n اعطاء القيمة 7 للمتغير
>>> msg = "Quoi de neuf ?"      # msg للمتغير (Quoi de neuf ?) تعيين القيمة
>>> pi = 3.14159                # pi اعطاء قيمة للمتغير
```

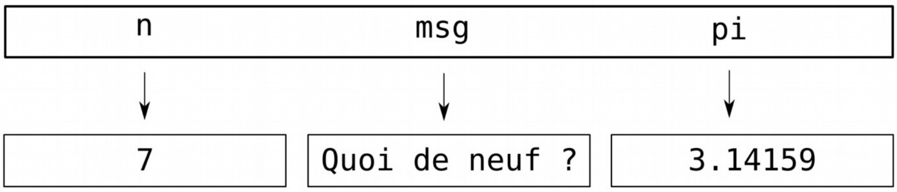
و توضح الأمثلة أعلاه تعيين البيانات في بيثون، بعد أن تم تنفيذها في ذاكرة الحاسوب، في أماكن مختلفة:

- أسماء ثلاثة متغيرات: **msg**، **n** و **pi**
- تسلسل البيانات الثلاثة، والتي يتم ترميز 7 بعدد صحيح وجملة **Quoi de neuf ?** بسلسلة والعدد 3,14159 بعدد صحيح.

البيانات الثلاثة المذكورة أعلاه للتعيين، كان لكل منها أثر في تنفيذ عدة عمليات في ذاكرة الحاسوب:

- إنشاء وتخزين اسم متغير.
- تعيين نوع محدد جيد (سيتم توضيحه في الصفحة التالية).
- إنشاء وتخزين قيمة معينة.
- ارتباط (عن طريق مؤشرات داخلية) بين اسم المتغير وموقع الذاكرة من القيمة المطابقة.

أفضل طريقة للشرح هي تمثيل كل هذا برسم تخطيطي:



أسماء المتغيرات الثلاثة مخزنة ولديها مراجع في منطقة معينة من الذاكرة تسمى مساحة الأسماء، في حين تقع القيم المناظرة في أماكن أخرى، وأحيانا بعيدة جدا عن بعضها البعض، وسوف نوضح هذا المفهوم أكثر في الصفحات القادمة.

عرض قيمة متغير

بعد التمارين المذكورة أعلاه، سيكون لدينا ثلاثة متغيرات **msg**، **n** و **pi** لعرض قيمة متغير على الشاشة، هنالك طريقتان، الأولى هي كتابة اسم المتغير ثم النقر على مفتاح الإدخال "Enter"، فسيستجيب بيثون فيعرض القيمة:

```
>>> n
7
>>> msg
'Quoi de neuf ?'
>>> pi
3.14159
```

هذه ميزة ثانية للمفسر، الذي يهدف إلى جعل الحياة أسهل عند القيام بتمارين بسيطة في سطر الأوامر، في داخل البرنامج، سوف تستخدم دائما print()[5]:

```
>>> print(msg)
Quoi de neuf ?
>>> print(n)
7
```

لاحظ الفرق الدقيق بين طرق العرض التي تم الحصول عليها مع كل طريقة. وظيفة print() لا تظهر بدقة قيمة المتغير كما أنه تم ترميزها، في حين أن الطريقة الأخرى (وهي فقط إدخال اسم المتغير) يعرض مثلا علامات الاقتباس لكي يذكركم أنكم تتعاملون مع متغير مثل سلسلة (ستتعرف إلى هذا في وقت آخر).

> في الإصدارات السابقة لبيثون. دور دالة الطباعة print() هو التعليمة print الخاصة (مما يجعلها من الكلمات المحجوزة). وهذه التعليمة تستخدم بدون أقواس. في التمارين السابقة. يجب عليك إذاً كتابة print n أو print msg. وإذا أردت لاحقا استخدام بيثون 3 في برامج مكتوبة بإحدى نسخ بيثون القديمة. يجب عليك إضافة الأقواس بعد كل تعليمة print لتحويلها إلى دالة (يمكن للأدوات المساعدة القيام بذلك تلقائيا).
>
> في هذه الإصدارات القديمة. يتم معالجة السلاسل النصية بشكل مختلف (سوف نتحدث عن هذا بالتفصيل لاحقا). واعتمادا على تكوين جهاز حاسوبك. سوف تواجه بعض الرموز الغريبة عندما تتعامل مع سلاسل نصية تحتوي على أحرف معلمة. مثل:
>
> ```
> >>> msg = "Mon prénom est Chimène"
> >>> msg
> 'Mon pr\xe9nom est Chim\xe8ne'
> ```
>
> هذا الأشياء الغريبة تنتمي إلى الماضي. ولكن سوف نرى لاحقا كيف أن المبرمج الجديد بهذا الاسم يجب أن يعرف كيف يتم ترميز المحارف التي تواجهه في مصادر مختلفة من البيانات. لأن تحديد هذه الترميزات تغيرت على مر السنين. ويجب أن نعرف التقنيات لتحويلها.

كتابة المتغيرات

في بيثون ليس من الضروري كتابة أسطر معينة من التعليمات البرمجية لتعريف نوع المتغير قبل استخدامها. لأنه ببساطة عند تعيين قيمة إلى اسم المتغير يقوم بيثون بوضع نوع المتغير تلقائيا مع النوع الذي يطابق القيمة المقدمة.

في التمرين السابق، على سبيل المثال، يتم إنشاء أنواع المتغيرات تلقائيا مثلا (**n**: صحيح، **msg**: سلسلة، **pi**: عدد حقيقي).

هذه هي ميزة مثيرة للاهتمام في بيثون، التي تتبع لعائلة معينة من اللغات حيث يوجد من نفس العائلة على سبيل المثال، ليسب وسكيم...إلخ، طريقة كتابة المتغيرات في بيثون حيوية على عكس الكتابة الثابتة التي هي على قاعدة ثابتة، على سبيل

[5] سيتم تحديد المهام بالتفصيل في الفصول 6 و 7 (انظر الصفحة 51).

المثال في لغة سي أو جافا يجب الإعلان أول مرة عن اسم ونوع المتغير، عندها فقط يمكنك تعيين المحتوى، والذي يجب بالطبع أن يكون متوافقا مع النوع المعلن.

الكتابة الثابتة هي الأفضل في حالة اللغات المترجمة؛ لأنه يحسن عملية الجمع (التي نتيجتها برنامج بالبيناري).

الطباعة الديناميكية أكثر سهولة لكتابة بنيات المنطقية لمستوى عال (الانعكاسية، متابروغراميك) ولا سِيَّما في سياق البرمجة الكائنية (تعدد الأشكال) كما يسهل استخدام هياكل البيانات الغنية مثل القوائم والقواميس.

تعدد المهام

في بيثون يمكنك تعيين قيمة لمتغيرات عدة في نفس الوقت، على سبيل المثال:

```
>>> x = y = 7
>>> x
7
>>> y
7
```

يمكنك أيضا إعطاء قيمة لكل متغير من المتغيرات في آن واحد معا:

```
>>> a, b = 4, 8.33
>>> a
4
>>> b
8.33
```

في هذا المثال، تم تعيين قيم مختلفة لكل من **a** و **b** -في وقت واحد- هي **4** و **8,33**.

> الفرنكوفونيين (والعرب أيضا) لديهم عادة باستخدام الفاصلة كفاصل عشري، وأما لغات البرمجة فتستخدم دائما الاتفاقية الحالية بين البلدان الناطقة باللغة الإنكليزية، وهذا معناه أن تستخدم النقطة العشرية. والفاصلة , التي نستخدمها نحن ، تستخدم لفصل مختلف العناصر (برامترات، إلخ) وكما رأينا في مثالنا، لقيم المتغيرات التي قمنا بتعيينها.

تمارين

1.2 وضح ماذا يحدث عند كتابة كل واحدة من التعليمات الثلاثة للمثال في الأسفل:

```
>>> largeur = 20
>>> hauteur = 5 * 9.3
>>> largeur * hauteur
930
```

2.2 قم بتعيين 3،5،7 إلى المتغيرات a،b،c. ثم قم بتنفيذ a-b//c وفسر النتيجة المتحصل عليها.

العوامل والتعابير

قم بالتلاعب بقيم المتغيرات مع المعاملات لتشكيل التعبيرات. على سبيل المثال:

```
a, b = 7.3, 12
y = 3*a + b/5
```

في هذا المثال، بدأنا بتعيين للمتغيرين **a** و **b** القيم 7,3 و 12.

كما شرحنا سابقا، يقوم بيثون تلقائيا بتعيين نوع "حقيقي" للمتغير **a**، والنوع "صحيح" للمتغير **b**.

السطر الثاني من مثالنا، يقوم بتعيين للمتغير الجديد y نتيجة التعبير الذي يحتوي على العوامل *، + و/ مع المعاملات **a**، **b**، **3** و 5. والعوامل هي رموز خاصة تستخدم لتمثيل العمليات الحسابية البسيطة مثل الجمع والطرح والمعاملات هي قيم تجمع بمساعدة العوامل.

يقوم بيثون بتقييم كل تعبير يقدم إليه، ولو كان معقد، ونتيجة هذا التقييم هو دائما قيمة، لهذه القيمة يتم تلقائيا تعيين نوعها، وهي تعتمد على ما يوجد في التعبير. في المثال أعلاه، سيكون y نوع حقيقي، لأن التعبير الذي تم تقييمه يحتوي على الأقل عدد حقيقي.

عوامل بيثون ليست فقط العوامل الأربعة الرياضية الأساسية. ولقد رأينا بالفعل وجود عامل القسمة الصحيح //. ويوجد أيضا العامل ** للأسس (القوة)، ويوجد العديد من العوامل المنطقية، مثل عوامل السلاسل النصية، وعوامل اختبار الهوية أو الانتماء وإلخ. سوف نتحدث على كل هذا في الصفحات القادمة.

وبالمناسبة يوجد عامل مودولو (modulo)، المُمَثَّل بالرمز %. هذا العامل يقوم بتوفير ما تبقى من عملية القسمة لعدد صحيح. حاول على سبيل المثال:

```
>>> 10 % 3        # (لاحظ ما يحدث!)
>>> 10 % 5
```

هذا العامل سيكون مفيدا في وقت لاحق، وخاصة لاختبار ما إذا كان العدد **a** يقبل القسمة على **b**. ويكفي للتحقق كتابة **a % b** ليعطي نتيجة صفر إذا كان يقبل القسمة.

تمارين

3.2 اختبر أسطر التعليمات، وصِف ما يحدث:

```
>>>  r , pi = 12, 3.14159
>>>  s = pi * r**2
>>>  print(s)
>>>  print(type(r), type(pi), type(s))
```

في رأيك، ما فائدة الدالة **type**()؟

(ملاحظة: سيتم وصف المهام بالتفصيل في الفصلين 6 و 7).

ترتيب المعاملات

عندما يكون هنالك أكثر من عامل في التعبير، فيجب علينا أن نعرف ترتيب العمليات التي تعتمد على القواعد الأولية، في بيثون، القواعد الأولية هي نفس قواعد الرياضيات التي كنت تدرسها، ويمكنك حفظها بسهولة باستخدام خدعة PEMDAS إختصارا لـ:

- *P* ما بين قوسين، ما بين القوسين له أولوية قصوى، حيث تستطيع أن ترتب التعبير بالترتيب الذي تريده. هكذا 2*(1-3) = 4، و(1+1)**(2-5) = 8.
- *E* للأسس (القوة)، الأولوية الثانية للأسس قبل غيرها من العمليات. هكذا 2**1+1 = 3 (ليس 4)، و 3*1**10 = 3 (ليس 59049!).
- *M* و *D* للضرب والقسمة، التي لهما نفس الأولوية. يتم تقيمهما قبل الجمع *A* والطرح *S* اللذان يجران آخر الأمر. هكذا 2*3-1 = 5 (بدلًا من 4)، و 1-2/3 = -0.3333... (بدلًا من 1.0).
- إذا كان هناك أكثر من عاملين لهما نفس الأولوية، يتم التقييم من اليسار إلى اليمين.
- وبالتالي في التعبير 59*100//60، يجب إجراء الضرب أولا، ثم ينفذ الجهاز 5900//60 الذي يعطي 98، إذا أجريت القسمة أولا ستكون النتيجة 59 (تذكر هنا أن المعامل // يقوم بعملية قسمة عدد صحيح، وقم بالتحقق من خلال 59*(100//60)).

البنية

حتى الآن اطلعنا على مختلف عناصر لغة البرمجة وهي: المتغيرات، التعبيرات والتعليمات ولكن دون معالجة، كيف يمكننا أن نوحدها بعضها مع بعض؟

نأتي إلى ذكر واحدة من نقاط القوة الكبيرة لِلُغات البرمجة عالية المستوى هي أنها تتيح بناء مجموعة تعليمات عن طرق تجميع أجزاء مختلفة، على سبيل المثال، إذا كنت تعرف كيفية إضافة رقمين وكيفية عرض قيمة، يمكنك الجمع بين هاتين التعليمتين:

```
>>> print(17 + 3)
>>> 20
```

هذا قليل من كثير، ستتوضح هذه الميزة عندما تقوم بخوارزميات معقدة بوضوح واختصار، على سبيل المثال:

```
>>> h, m, s = 15, 27, 34
>>> print("عدد الثواني المنقضية منذ منتصف الليل = ", h*3600 + m*60 + s)
```

تنبيه: هناك حد لما يمكن جمعه:

في العبارة، يجب عليك أن تضع اسم المتغير على الجانب الأيسر من علامة المساواة وليس التعبير وذلك لأنها لا تمتلك نفس المعنى كما في الرياضيات: كما لاحظنا في وقت سابق وهذه هي مهمة هذا الرمز (وضع بعض المحتويات إلى متغير) وهو ليس رمزًا للمساواة. وسوف نتحدث على رمز المساواة (على سبيل المثال في الجمل الشرطية) فيما بعد.

على سبيل المثال، العبارة `m + 1 = b` خاطئة.

من السلبيات، كتابة `a = a + 1` لأنه أمر غير مقبول في الرياضيات، في حين أن هذا النوع من الكتابة شائع جدا في البرمجة. معنى `a = a + 1` هو زيادة قيمة المتغير بقيمة 1.

سوف نتحدث حول هذا الموضوع قريبا. نحن نحتاج أولا إلى فهم مفهوم أكثر أهمية.

3

التحكم في تلقيم التنفيذ

في الفصل الأول، عرفنا أن النشاط الأساسي للمبرمج هو إيجاد حلول للمشاكل، ومن أجل حل مشكلة في الحاسوب، يجب دائما القيام بسلسلة من الإجراءات بترتيب معين، ويطلق على هذه الإجراءات والترتيب الذي ينبغي القيام به بالخوارزمية.

في بيثون يسمى هذا البرنامج بتلقيم التنفيذ، وتسمى الهياكل التي تعدلها بتعليمات التحكم في التلقيم ببنية التحكم.

وهي مجموعات من التعليمات التي تحدد ترتيب الإجراءات التي يتم تنفيذها. وفي البرمجة الحديثة، توجد ثلاثة أنواع فقط: السلسلة والشروط -التي سنناقشها في هذا الفصل- والتكرار الذي سنناقشه في الفصل القادم.

تسلسل التعليمات

الذي لم نذكره هو أنه يتم تنفيذ تعليمات البرنامج الواحدة تلو الأخرى بالترتيب كما هي مكتوبة في السكربت.

قد يبدو لك هذا تافها للوهلة الأولى، ومع ذلك فإن التجربة تدل على أن عددا كبيرا من الأخطاء الدلالية في برامج الحاسوب هي نتيجة لتعليمات سيئة (خاطئة). كلما تقدمت أكثر في فن البرمجة، سوف تدرك أكثر حول الحذر بشأن المكان الذي يجب وضع التعليمات الخاصة بك واحدة تلو الأخرى. على سبيل المثال، في تسلسل التعليمات التالية:

```
>>> a, b = 3, 7
>>> a = b
>>> b = a
>>> print(a, b)
```

سوف تحصل على نتيجة عكسية إذا كنت قد بدلت بين السطر الثاني والثالث.

بيثون يدير التعليمات في الوضع الطبيعي من البداية إلى النهاية، إلا إن واجه جملا شرطية مثل **if** كما هو واضح أدناه (سوف نتعرف على غيرها ولا سِيَّما الحلقات التكرارية)، وهذه التعليمات تسمح للبرنامج بمتابعة المسارات المختلفة تبعا للظروف.

تحديد أو تنفيذ شرط

إذا كنا نريد كتابة تطبيقات مفيدة، فنحن بحاجة إلى توجيه تقنيات تشغيل البرنامج في اتجاهات مختلفة، تبعا للظروف التي تواجه البرنامج، وللقيام بذلك نحن بحاجة إلى تعليمات لاختبار شرط وتعديل سلوك البرنامج وفقا لذلك.

أبسط هذه الجمل الشرطية هي التعليمة **if**، ولاختبار عملها يجب عليك إدخال هذين السطرين إلى محرر بيثون:

```
>>> a = 150
>>> if (a > 100):
...
```

الأمر الأول هو لتعيين قيمة 150 إلى المتغير **a**. حتى الآن لا شيء جديد.

و عند الانتهاء من إدخال السطر الثاني، ستجد أن بيثون سيتفاعل بطريقة جديدة، في الواقع إذا لم تنسَ الرمز ":" في نهاية السطر، ستجد أنه قد تم استبدال الموجه الرئيسي (>>>) إلى موجه ثانوي يتكون من ثلاثة نقاط.

إذا كان لديك محرر لا تلقائي، يجب عليك الآن إدخال تبويت (أو أدخل أربعة مسافات) قبل الدخول إلى السطر التالي، بحيث يبدأ بذلك، وينبغي أن تظهر الشاشة على النحو التالي:

```
>>> a = 150
>>> if (a > 100):
...     print("a dépasse la centaine")
...
```

انقر مرة أخرى زر الإدخال (Enter) سترى أن البرنامج سيعمل وستحصل على:

```
a dépasse la centaine
```

كرر نفس العملية لكن مع `a = 20` في السطر الأول: في هذه المرة، بيثون لم يظهر شيئًا.

التعبير الذي وضعته بين قوسين نسميه شرط، تُستخدَم `if` لاختبار صحة هذا الشرط، فإذا كان الشرط صحيحا فسوف يتم تنفيذ ما بعد الرمز ":" وإذا كان الشرط غير صحيح، لا يحدث شيء، لاحظ أن الأقواس المستخدمة هنا مع عبارة **if** اختيارية، لقد استخدمناها لتحسين إمكانية القراءة، في اللغات الأخرى، قد تصبح إلزامية.

أكرر مرة أخرى، بعد إضافة أول سطرين كما هو مبين في الأسفل. تأكد من أن السطر الرابع بدأ على اليسار (بدون مسافة البادئة)، ولكن مرة أخرى نضيف بادئة جديدة في السطر الخامس (ويفضل نفس مسافة بادئة السطر الثالث):

```
>>> a = 20
>>> if (a > 100):
...     print("a dépasse la centaine")
... else:
...     print("a ne dépasse pas cent")
...
```

أنقر مرة أخرى زر الإدخال (Enter) سيعمل البرنامج وسيعرض:

```
a ne dépasse pas cent
```

كما فهمت أنت، العبارة **else** "إذا" تسمح للمبرمج بتعيين تنفيذ بديل في برنامج من احتمالين. ويمكننا فعل أفضل من هذا عن طريق استخدام العبارة **elif** (دمج else if):

```
>>> a = 0
>>> if a > 0 :
...     print("a est positif")
... elif a < 0 :
...     print("a est négatif")
... else:
...     print("a est nul")
...
```

مقارنة المعاملات

التعليمة **if** للتحقق من الشروط، قد تحتوي على عوامل المقارنة التالية:

```
x == y       # x يساوي y
x != y       # x لا يساوي y
x > y        # x أكبر من y
x < y        # x أصغر من y
x >= y       # x أكبر من أو يساوي y
x <= y       # x أصغر من أو يساوي y
```

مثال

```
>>> a = 7
>>> if (a % 2 == 0):
...     print("a est pair")
...     print("parce que le reste de sa division par 2 est nul")
... else:
...     print("a est impair")
...
```

لاحظ أن عامل المساواة بين القيمتين يتكون من رمزيّ مساواة وليس واحدًا. علامة مساواة واحدة هي عامل تعيين وليس عامل مقارنة. وسوف تجد نفس الرمز في لغات أخرى كجافا وسي بلس بلس.

بيانات المشغّل - عبارة الكتل

البناء الذي استخدمته مع العبارة **if** هو مثالك الأول من مشغّل البيانات. قريبا سوف تجمع آخرين. في بيثون، البيانات المركبة لديها نفس البنية: نقطتان عموديتان تليهما عبارة أو أكثر البادئة تحت خط العمود، على سبيل المثال:

```
Ligne d'en-tête:
    première instruction du bloc
    ... ...
    ... ...
    dernière instruction du bloc
```

إذا كان هنالك عبارات متعددة مسبوقة بِبادئة تحت الخط العمودي، يجب أن يكونوا على نفس المستوى (على سبيل المثال 4 مساحات فارغة)، بادئة هذه التعليمة هي ما نسميه الآن كتلة العبارات. كتلة العبارات هي سلسلة من التعليمات تُشكِّل وِحدة المَنطق، والتي لا يتم تنفيذها إلا في ظروف معينة يتم تحديدها في الخط العمودي. في المثال في الفقرة السابقة، سطريّ العبارة تحت السطر الذي يحتوي العبارة **if** هما كتلة المنطق نفسها فسيتم تنفيذ هذين السطرين (على حد سواء) إذا كان اختبار الشرط مع العبارة صحيحا وهذا معناه باقي القسمة على 2 لا شيء.

التداخلات

من الممكن أن تتداخل في كل المركبات عدة تصاريح أخرى من أجل تنفيذ المشغّل هياكل صنع القرار، على سبيل المثال:

```
if embranchement == "vertébrés":                   # 1
    if classe == "mammifères":                     # 2
        if ordre == "carnivores":                  # 3
            if famille == "félins":                # 4
                print("c’est peut-être un chat")   # 5
        print("c’est en tous cas un mammifère")    # 6
    elif classe == "oiseaux":                      # 7
        print("c’est peut-être un canari")         # 8
print("la classification des animaux est complexe") # 9
```

تحليل هذا المثال، هذا الجزء من البرنامج لا يطبع عبارة " c’est peut-être un chat " إلا إن كانت نتائج اختبار أول أربعة شروط صحيحة.

ليتم عرض جملة "c’est en tous cas un mammifère" يجب أن يتحققا شرطان. العبارة المطبوعة في هذه الجملة (السطر الرابع) هي في الواقع على مستوى المسافة البادئة لنفس التعليمات: **"if ordre == "carnivores** (السطر الثالث) وبالتالي هما -على حد سواء- جزء من الكتلة نفسها، والتي تُعرض إذا كانت نتائج اختبار الشروط في السطرين الأول والثاني صحيحة.

ليتم عرض عبارة "c’est peut-être un canari" يجب أن يكون المتغير **embranchement** يحتوي على "vertébrés"، ومتغير **classe** يحتوي على "oiseaux".

أما بالنسبة للجملة في السطر التاسع، فيتم عرضها في جميع الحالات، لأنها جزء من كتلة نفس البيانات في السطر الأول.

بعض القواعد لبناء جملة في بيثون

كل ما سبق يقودنا إلى بعض قواعد بناء جملة :

تعريف حدود التعليمات والكتل بالتخطيط

في الكثير من لغات البرمجة، يجب عليك إتمام كل سطر من التعليمات برمز خاص (غالبا ما يكون الفاصلة المنقوطة)، في بيثون، هذا الحرف في نهاية السطر يلعب هذا الدور، (سنرى لاحقا كيفية استعمال هذه القاعدة لتوسيع مشغّل البيانات إلى أسطر متعددة) وأيضا إستكمال خط التعليمة مع التعليق. تعليق بيثون يبدأ دائما مع الحرف الخاص # ويتم تجاهل كل ما يتم تضمينه والانتقال إلى السطر التالي من قِبَل المشغّل.

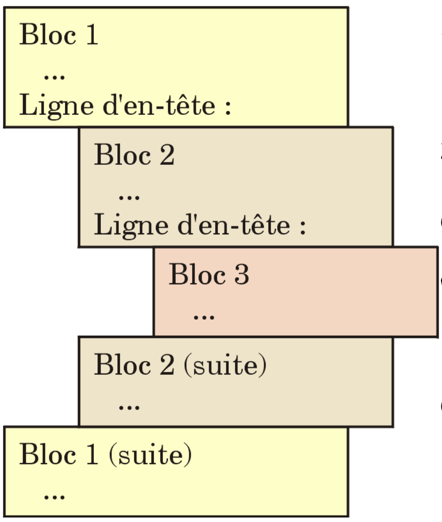
و في معظم لغات البرمجة الأخرى، يجب أن تكون كتلة البيانات مع رموز معينة (حتى في بعض الأحيان تعليمات البداية **begin** والنهاية **end**). في سي بلس بلس وجافا على سبيل المثال، يجب وضع كتلة البيانات داخل أقواس، وهذا يسمح لكتابة كتل من التعليمات واحدة تلو الأخرى، من دون قلق أو فواصل أو سطر المسافة البادئة ولكن هذا يمكن أن يؤدي إلى ارتباك في كتابة البرامج، فنحن البشر من الصعب علينا قراءة هذه الفقرات. ولذلك ينصح جميع المبرمجين الذين يستخدمون هذه اللغات باستخدام فواصل الأسطر والمسافات البادئة لترسم كتلا بصرية.

مع بيثون، يجب استخدام فواصل الأسطر والمسافات البادئة، ولكن في المقابل لا يوجد ما يدعو للقلق من رموز كتلة التحديدات الأخرى. في النهاية، بيثون يدفعك لكتابة شفرة مصدرية قابلة للقراءة وأخذ العادات الجيدة التي كنت تحتفظ بها عند استخدام لغات أخرى.

مشغّل البيانات: الرأس، النقطتان وكتلة بادئة للبيانات

سيكون لدينا فرصا عديدة لاستكشاف مفهوم "تعليمة الكتلة" والقيام بتمارين حول هذا الموضوع في الفصل التالي.

و يلخص الرسم البياني أدناه المبدأ.



- ترتبط التعليمات دائما مع الخط العمودي الذي يحتوي على تعليمات محددة للغاية (def ، while، else ، elif، if ...إلخ) التي تنتهي بنقطتين .
- الكتل محددة من قبل مسافة البادئة: يجب أن تكون جميع الأسطر على طريقة (طول البادئة) (و هذا يعني التحول إلى نفس العدد من المسافات) يمكن استخدام أي عدد للمسافات ولكن معظم المبرمجين يستخدمون مضاعفات الرقم 4.
- لاحظ أنه يمكن للكود الكتابة بعيدا (كتلة 1) في حد ذاتها يمكن إزالتها من الهامش الأيسر (تداخل في أي شيء)

> **مهم**
>
> يمكنك أيضا وضع مسافة البادئة من خلال علامات التبويت، ولكن يجب أن تكون بعد ذلك حذرا جدا بعدم استخدام المسافات في بعض الأحيان، وفي بعض الأحيان الأخرى مسافات البادئةلتعريف أسطر نفس الكتلة. وحتى لو تبدو مطابقة على الشاشة، فالمسافات وعلامات التبويت هي رموز ثنائية منفصلة : ولذلك ينظر بيثون إلى هذه الأسطر على أنها جزء من كتل مختلفة. والذي قد يؤدي إلى أخطاء يصعب تصحيحها. ولذلك يفضل معظم المبرمجين استخدام علامات التبويت. إذا كنت تستخدم محرر نصوص "ذكي"، فمن الأفضل تفعيل خيار "استبدال علامات التبويت بمسافات" .

المسافات والتعليقات عادة ما يتم تجاهلها

بصرف النظر عن تلك المستخدم لمسافة البادئة (في بداية السطر)، المساحات الموضوعة داخل التعليمات والتعبيرات دائما ما يتم تجاهلها (ما لم تكن جزء من سلسلة نصية). وهذا نفس الشيء بالنسبة للتعليقات: فهي تبدأ برمز # وتمتد إلى نهاية السطر الحالي (يتم تجاهلها).

4

تعليمات التكرار

من المهام التي تبذل فيه الآلات قصارى جهدها هي تكرار المهام المتماثلة من دون خطأ. هنالك العديد من الطرق لبرمجة هذه المهام المتكررة. سنبدأ مع واحدة تعتبر الأكثر أساسية وهي حلقة تكرار *while*.

إعادة التعيين

حتى الآن لم نخبرك بأنه يجوز إعادة تعيين قيمة جديدة لمتغير واحد، مرة واحدة أو عدة مرات على النحو المطلوب.

عندما نقوم بإعادة تعيين، نحن نقوم باستبدال القيمة القديمة بقيمة جديدة لنفس المتغير .

```
>>> altitude = 320
>>> print(altitude)
320
>>> altitude = 375
>>> print(altitude)
375
```

هذا يجعلنا نلفت انتباهكم مرة أخرى إلى أنه يجب علينا أن لا نخلط بين رمز التعيين في بيثون ورمز المساواة كما يفهم في الرياضيات ، لأن هنالك العديد من الأشخاص يفسرون هذه العبارة **altitude = 320** كأن هذا الرمز للمساواة، وهي ليست كذلك!

- أولا المساواة هي عملية تبادلية، في حين أن التعيين ليس كذلك فمثلا في الرياضيات كتابة **a = 7** أو **a** = 7 نفس الشيء في حين أن في البرمجة (على سبيل المثال **altitude** = 375) سيكون غير منطقي.
- ثانيا، المساواة تكون دائمة، في حين يمكن استبدال قيمة المتغير في بيثون باستخدام رمز التعيين كما رأينا للتو. في الرياضيات نؤكد أن هذه مساواة أ = ب في بداية البرهان، ثم نواصل لتكون مساوية لـ ب في جميع التطورات القادمة . في البرمجة، عبارة التعيين الأولى هي معادلة قيمتين، والعبارة الأخرى لاستبدال قيمة الأولى، على سبيل المثال:

```
>>> a = 5
>>> b = a          # لديهم نفس القيمة a و b
>>> b = 2          # أصبحوا مختلفان الآن a و b
```

تذكروا أن بيثون تسمح لك بتعيين عدة قيم متغيرات في نفس الوقت :

```
>>> a, b, c, d = 3, 4, 5, 7
```

هذه ميزة من ميزات بيثون الأكثر إثارة للاهتمام عند النظرة الأولى.

على سبيل المثال، لنفترض أننا صنعنا المتغيرين **a** و **c** وهما يحتويان على القيمتين **3** و **5** نريد عكس القيمتين) كيف نفعل هذا؟

تمرين

1.4 اكتب الكود اللازم لتحقيق هذه النتيجة (فوق).

بعد القيام بالعملية المقترحة أعلاه، سوف تجد بالتأكيد وسيلة، وسيطلب منك المعلم على الأرجح التعليق في الصف. لأن هذه هي عملية مشتركة، إن لغات البرمجة غالبا ما تقدم اختصارات لأداء (التعليمات المتخصصة على سبيل المثال، مثل تعليمة المبادلة الأساسية) في بيثون، يمكن تعيين مساوي في البرمجة التبادل بشكل أنيق وخاص:

```
>>> a, b = b, a
```

(ويمكن بالطبع عكس متغيرات أكثر (مثلا ثلاثة) في نفس الوقت بنفس التعليمة)

حلقات التكرار - العبارة while

في البرمجة، نسمي حلقة التكرار بنظام التعليمة الذي يكرر المهمة المحددة عدة مرات (أو بشكل لا نهائي) جميع العمليات .بيثون يقترح عبارتين لاستعمال الحلقات: العبارة **in … for** … قوية جدا وسوف ندرسها في الفصل العاشر والبيان **while** الذي سندرسه الآن.

الرجاء ادخال الأوامر التالية:

```
>>> a = 0
>>> while (a < 7):          # (لا تنسَ النقطين!)
...     a = a + 1           # (لا تنسَ مسافة البادئة!)
...     print(a)
```

اضغط مرة أخرى على Enter.

ماذا حدث؟

قبل قراءة التعليقات التالية، خذ وقتا لفتح دفتر الملاحظات ودون هذه السلسلة من الأوامر. وصف أيضا نتيجة ذلك وحاول تفسر كيف حدث هذا بأكبر قدر ممكن من التفصيل.

التعليقات

تعني كلمة **while** بالإنكليزية "عندما". هذه العبارة المستخدمة في السطر الثاني تخبر بيثون بأنه يجب عليه تكرار الكتلة التالية من البيانات باستمرار، عندما يكون المتغير **a** أقل من 7.

مثل عبارات **if** والتي ناقشناها في الفصل السابق، في حين تبدأ العبارة **while** بعبارة المجمع. النقتطين العاموديتين في نهاية السطر تخبر بيثون أنه سيبدأ بدخول مجمع يجب تكراره والذي يجب أن يبدأ بمسافة. كما تعلمت في الفصل السابق يجب أن تكون بادئة جميع البيانات داخل الكتلة متساوية بالضبط في نفس المستوى (وهذا يعني البدء بالعدد نفسه من المسافات).

لقد قمنا ببناء حلقتنا الأولى، التي تكرر كتلة البادئة للبيانات عدة مرات، ولكن كيف تعمل:

- مع العبارة **while** يبدأ بيثون بالتأكد من الشرط الموضوع داخل القوسين (القوسين اختياري ونحن استخدمناها للتوضيح فقط).
- إذا كان الشرط غير صحيح، يتم تجاهل الكتلة كلها ويتم إنهاء البرنامج[6].
- إذا كان الشرط صحيحا، يقوم بيثون بتنفيذ كتلة البايانات والتي تشكل هيئة حلقة وهذا معناه:
  - التعليمة **a = a + 1** معناها زيادة واحد إلى قيمة المتغير (وهذا معناه إعادة تعيين قيمة لمتغير بقيمته القديمة زائد واحد).
  - استدعاء دالة الطباعة **print()** لإظهار قيمة الحالية للمتغير **a**.
- وعند تنفيذ هاتين التعليمتين يتم الرجوع مرة أخرى إلى عبارة التكرار، وهذا معناه الرجوع إلى عبارة التكرار وإجراء الشرط إذا كان صحيحا يكمل وإذا كان غير صحيح يخرج.

  في مثالنا هذا، إذا كان الشرط **a < 7** لايزال صحيحا، سيتم تنفيذ الحلقة مرات أخرى وتستمر الحلقة.

ملاحظات

- يجب أن يكون المتغير الذي يتم اختباره موجودًا بالفعل (أي أنه يجب صنع المتغير ويجب أن يكون يحتوي على قيمة مثلا: 1)
- إذا كان الشرط غير صحيح من البداية، لن يتم تنفيذ ما داخل الحلقة أبدا (الكتلة)
- إذا كان الشرط صحيحا دائما، يتم تكرار الحلقة إلى مالا نهاية (عندما تكون بيثون يعمل) .لذا يجب علينا أن نضع في الكتلة حلقة واحدة على الأقل تغير قيمة المتغير الذي يؤثر على الحلقة في الوقت المناسب (حتى يصبح هذا الشرط غير

---

6 على الأقل في هذا المثال. سوف نرى فيما بعد أن التنفيذ يستمر مع أول تعليمة تلي كتلة البادئة, والتي هي جزء من نفس كتلة العبارة **while** نفسها .

صحيح وتنتهي الحلقة).

مثال على الحلقة اللانهائية (تجنبها !):

```
>>> n = 3
>>> while n < 5:
...     print("hello !")
```

تطوير الجداول

أعد كتابة التمرين الأول مع تغيير طفيف أدناه:

```
>>> a = 0
>>> while a < 12:
...     a = a +1
...     print(a , a**2 , a**3)
```

يجب عليك أن تحصل على قائمة من المربعات والمكعبات من 1 إلى 12.

اعلم أنك تستطيع أن تمرر أكثر من برامتر في دالة الطباعة **print()** لإظهار عدة تعبيرات في نفس الوقت واحدة تلو الأخرى على نفس الخط: فقط ضع فاصل بين كل واحدة وأخرى. بيثون سيضع مسافة بين العناصر المعروضة.

البناء الرياضي

البرنامج الصغير الذي بالأسفل يعرض أول عشرة أرقام من تسلسل يسمى بـ "تسلسل فيبوناتشي". هذه سلسلة من الأرقام كل رقم من هذه السلسلة يساوي مجموع رقمين سابقين من نفس السلسلة. جرب تحليل هذا البرنامج (الذي يستخدم التعيين الموازي) وصف أكبر قدر ممكن دور كل سطر من التعليمات.

```
>>> a, b, c = 1, 1, 1
>>> while c < 11 :
...     print(b, end =" ")
...     a, b, c = b, a+b, c+1
```

عندما تشغل البرنامج ستحصل على ما يلي:

```
1 2 3 5 8 13 21 34 55 89
```

يتم عرض تسلسل فيبوناتشي على نفس السطر. هذا ما سنفهمه حتى البارامتر الثاني end =" " التي بها دالة الطباعة. افتراضيا، دالة الطباعة **print()** تقوم بإضافة رمز خاص لا يظهر ليقوم بالقفز إلى السطر التالي عندما نقوم بعرض شيء ما. والبارامتر end =" " يخبر بيثون أن يستبدل القفزة بمسافة صغيرة. إذا حذفت هذا البارامتر، سيتم عرض الأرقام واحدا تحت الآخر.

في برامجك المستقبلية، سوف تضع في برامجك في كثير من الأحيان حلقات تكرارية كما التي حللنها هنا. وسيتبادر إلى ذهنك سؤال أساسي، هل ستتعلم البرمجة بإتقان. تأكد أنك ستصل إلى ذلك تدريجيا، بفضل التمارين.

عندما تختبر مشكلة من هذا النوع، يجب عليك النظر إلى أسطر التعليمات، بطبيعة الحال، لكن انظر خاصة في المتغيرات المختلفة المشاركة في الحلقة. هذا ليس سهلا دائما، على العكس ذلك لمساعدتك على النظر بوضوح، بدون تكبد مشقة رسم جدول على الورق وضعنا في الأسفل جدولا يشرح برنامجنا "سلسلة فيبوناتشي":

المتغيرات	a	b	c
القيم الأولية	1	1	1
القيم التي تعيينها خلال التكرارات	1 2 3 5 ...	2 3 5 8 ...	2 3 4 5 ...
عبارة الاستبدال	b	a+b	c+1

في هذا الجدول، الذي صنعناه إلى حد ما "بأيدينا"، مشيرا سطرا بسطر المتغيرات التي تأخذ كل واحدة من هذه المتغيرات كما في التكرارات القادمة. سنبدأ بالكتابة في أعلى الجدول أسماء المتغيرات المعنية. في السطر التالي، القيم المبدئية لهذه المتغيرات (القيم قبل بداية الحلقة). أخيرا في الجزء السفلي من الجدول، نضع التعبيرات المستخدمة داخل الحلقة لتغير قيمة كل متغير في كل تكرار.

املأ بضعة الأسطر المقابلة للتكرار الأول. لصنع متغير لسطر، فقط طبق ما فعناه في الأسطر السابقة، والتعبير عن الاستبدال موجود أسفل كل عمود. افحص واحصل على نتيجة البحث. إذا لم تكن في هذه الحالة جيدة، يجب عليك البحث عن تعبير آخر.

تمارين

2.4 اكتب برنامجا يعرض أول 20 نتيجة لعملية الضرب في 7.

3.4 اكتب برنامجا لعرض جدول لتحويل اليورو إلى الدولار الكندي، تبدأ بالزيادة إلى الجدول ستكون "هندسية"، مثل التالي:

1 يورو = 1.65 دولار

2 يورو = 3.30 دولار

4 يورو = 6.60 دولار

8 يورو = 13.20 دولار

إلخ. (توقف عند 16384 يورو.)

4.4 اكتب برنامجا يعرض 12 رقما كل واحد من هذه الأرقام يساوي 3 مرات الرقم الذي قبله.

سكريبتك الأول. أو كيفية حفظ برامجنا

حتى الآن، نحن نستخدم بيثون في وضع تبادلي (وهذا معناه أنه كل مرة أدخلت الأوامر لم يتم حفظها). وهذا يسمح لك بتعلم أساسيات اللغة بسرعة، من خلال التجارب مباشرة هذه الطريقة لديها عيب واحد كبير : كل التعليمات التي تكتبها تختفي عند إغلاق المفسر. قبل مواصلة دراستك، حان الوقت لتعلم كيفية حفظ برامجك في ملفات على القرص الصلب أو مفتاح يو أس بي (USB) ، بطريقة تستطيع بها إعادة عمل المراحل ، ونقلها على حواسيب أخرى ، ...إلخ

للقيام بذلك، سوف تكتب الآن التعليمات الخاصة بك على أي محرر (مثل Kate و Geany و Gedit ...في لينكس و wordpad و Komodo و Geany ...على ويندوز، أو اكتب في واجهة التطوير الرسومية IDLE التي توزع مع بيثون في ويندوز)

و هكذا تكتب السكريبت، وثم تستطيع حفظه وتغيره ونقله وإلخ ... مثل أي مستند آخر مكتوب بالحاسوب.

بعدها، عندما ترغب في اختبار تنفيذ البرنامج الخاص بك ، يجب عليك فتح مفسر بيثون واكتب (اكتب كأنه بارامتر) اسم الملف الذي يحتوي على السكريبت. على سبيل المثال، إذا كتبت السكريبت داخل ملف يدعى "'MyScript"، يكفي أن تكتب هذا الأمر ليشتغل :

```
python3 MonScript 7
```

من الأفضل أن تتأكد من أن اسم ملف البرنامج ينتهي بـ **.py**

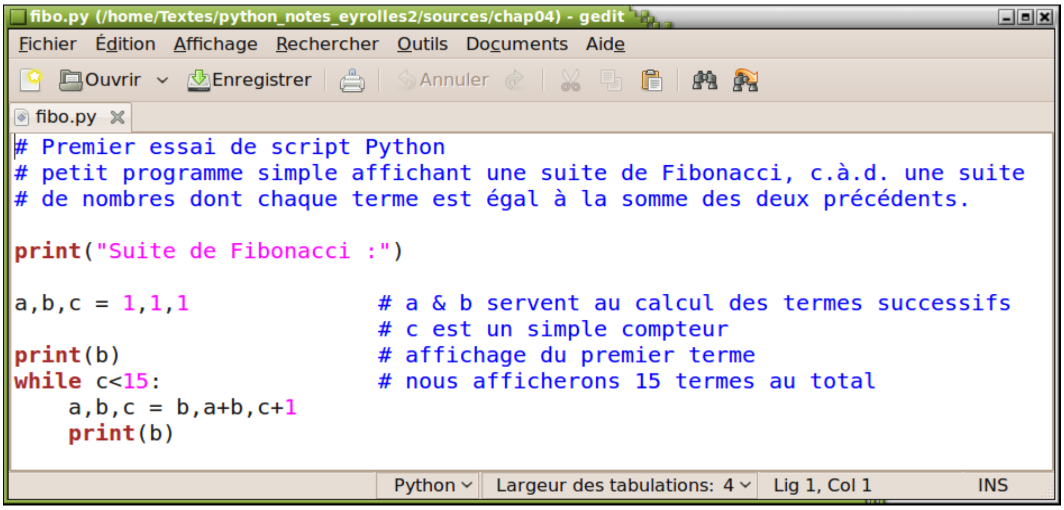
إذا اتبعت هذه النصيحة ، يمكنك تشغيل الملف النصي لبرنامج ، وذلك ببساطة عن طريق النقر على اسمه أو رمزه في مدير الملفات(و هو Explorer في ويندوز، أو Nautilus أو Konqueror على لينكس ...)

مديرو الملفات هؤلاء يعرفون أنهم يفتحون بيثون مع أي ملف ينتهي بـ **.py** (و هذا بالطبع يجب أن يكون قد تم تكوينها بشكل صحيح). نفس الشيء مع المحررات "الذكية" التي تعرف تلقائيا سكريبتات بيثون وتتكيف مع بناء التعليمات.

الشكل التالي يوضح استخدام محرر Gedit على لينكس "أبنتو" لكتابة السكريبت :

---

7 إذا تم تثبيت بيثون 3 على جهازك كمفسر بيثون افتراضي. يجب أن تكون قادرا على أن تكتب ببساطة : **python MonScript**. لكن انتبه : إذا كانت هنالك إصدارات متعددة من بيثون مثبتة على جهازك. ربما سيتم استخدام إصدار سابق من بيثون (الإصدار 2) .



سكريبت بيثون يحتوي على سلسلة تعليمات مماثلة لتلك التي اختبرناها الآن. وسوف تخزنها وبعد مدة سوف تشغلها وتقرأها من قبلك أو من قبل الآخرين ، وينصح وبشدة توضيح النصوص الخاصة بك إلى أقصى حد ممكن ، ويجب أن تتضمن الكثير من التعليقات ، والصعوبة الحقيقية في تطوير البرامج هي الخوارزميات الصحيحة ، بحيث يمكن التأكد من هذه الخوارزميات وتصحيحها وتغييرها وما إلى ذلك ، في أفضل ظروف ، ومن الضروري أن يصف المبرمج الكلمات جيدا وبأكبر قدر ممكن من الوضوح .

> المبرمج الجيد يدخل دائما أكبر عدد من التعاليق في سكريبتاته. وبذلك، لايسهل فهم الخوارزميات للقراء المحتملين الآخرين فقط. ولكنه يفرض أن سكريبته يكون أكثر وضوح .
> و أفضل مكان لهذا الوصف هو في جسم السكريبت (بحيث لا يضيع)

يمكننا إدراج تعليقات من أي نوع في أي مكان تقريبا في البرنامج النصي. ببساطة ضع قبل التعليق الرمز # حيث يعتبر المفسر هذا الرمز هو دلالة على تجاهل كل ما يأتي بعد هذا الرمز إلى نهاية السطر.

يرجى منك أن تفهم أنه يجب عليك أن تضع التعليقات كلما تقدمت في عملك في البرمجة. لا تنتظر حتى تنتهي من السكريبت الخاص بك. عليك أن تدرك أن المبرمج ينفق الكثير من الوقت لقراءة التعليمات البرمجية الخاصة بك (على سبيل التغيير والبحث عن الأخطاء ، إلخ ...) وسيكون هذا سهلا إذا وضعت العديد من التعليقات والملاحظات والتفسيرات.

افتح المحرر النصي ، واكتب السيناريو التالي :

```
# أول اختبار لسكريبت بيثون.
# برنامج صغير وبسيط يعرض تسلسل فيبوناتشي.
# كل عدد في هذه السلسلة يساوي مجموع اثنين سابقين .

a, b, c = 1, 1, 1            # تستخدم b و a لحساب الأعداد المتتالية
```

```
                                 # c هو عداد بسيط
print(b)                         # عرض العدد الأول
while c<15:                      # سوف نعرض 15 عدد
    a, b, c = b, a+b, c+1
    print(b)
```

لنوضح لكم الآن هذا المثال جيدا ، نحن بدأنا هذا السكريبت بثلاثة أسطر من التعليقات، التي تحتوي على وصف سريع لعمل البرنامج. اجعل هذه عادة، بفعل نفس الشيء في سكريباتاتك.

يجب أن تكون الأسطر البرمجية موثقة تماما. إذا فعلت مثلما فعلنا نحن هنا ، وهذا معناه أنك وضعت التعليقات بجانب الأسطر البرمجية ، تأكد أنك أزلت ما يكفي لكي لا تتداخل مع القراءة.

عندما تريد التأكد من نصك، احفظ البرنامج وشغله .

> و على الرغم من أن هذا ليس ضروريا. نحن نقترح عليك اختيار أسماء الملفات النصية التي تنتهي بـ ***.py***
> لأنه سوف يساعدك كثيرا على التعرف إليه. ومدراء الملفات الرسومية (مستكشف ويندوز، نوتيلوس وكونكيورر) تستخدم هذا الامتداد لوضع أيقونة معينة. ويجب عليك أيضا تجنب اختيار الأسماء التي هي أسماء وحدات بيثون بالفعل : مثل أسماء ***math.py*** أو ***tkinter.py*** !

إذا كنت تعمل في الوضع النصي في لينكس أو في نافذة دوس تستطيع تشغيل سكريبتك بمساعدة الأمر **python3** بالإضافة لاسم السكريبت. إذا كنت تعمل في الوضع الرسومي في لينكس ، تستطيع فتح نافذة الطرفية وفعل نفس الشيء :

```
python3 monScript.py
```

في مدير الملفات الرسومي ، يمكنك البدء في تشغيل البرنامج النصي بالضغط على أيقونته. لكن هذا لا يمكن إلا إذا كان بيثون 3 تم تعيينه كالمفسر الافتراضي للملفات التي تنتهي بـ **py.** (قد تحدث مشاكل في الواقع إذا كان هناك عدة إصدارات من بيثون مثبتة على جهازك. راجع المعلم لمزيد من التفاصيل).

إذا كنت تعمل مع IDLE ، يمكنك تشغيل البرنامج النصي الذي كتبته ، مباشرة باستخدام Ctrl + F5. في بيئات عمل أخرى متخصصة لتحديد بيثون، مثل Geany ستجد رموز أو اختصار للوحة المفاتيح لتشغيل البرنامج(و هذا يكون غالبا المفتاح F 5). استشر شخصا أو معلما عن طرق التشغيل الأخرى على أنظمة تشغيل مختلفة .

مشاكل محتملة مع الرموز

إذا قمت بكتابة السكريبت الخاص بك مع المحرر الأخير (مثل التقارير التي لدينا)ينبغي أن يعمل السيناريو المذكور أعلاه بدون مشاكل مع الإصدار الحالي مع بيثون 3. إذا كان البرنامج قديما أو تم تكوينه بشكل غير صحيح ، قد تحصل على رسالة مشابه لهذه :

```
File "fibo2.py", line 2
```

```
SyntaxError: Non-UTF-8 code starting with '\xe0' in file fibo2.py on line 2, but no
encoding declared; see http://python.org/dev/peps/pep-0263/ for details
```

هذه الرسالة تشير إلى أن السكريبت يحتوي على أحرف مطبعية مشفرة وفقا لمعايير قديمة (ربما ISO-8859-1 أو Latin-1).

نحن سنشرح مقاييس الترميز المختلفة في وقت لاحق في هذا الكتاب ، في الوقت الراهن ، اعلم أنك ستحتاج في هذه الحالة :

- إما إعادة تكوين محرر النص بحيث يتم الترميز برموز utf-8 (أو الحصول على محرر يعمل بهذه المعايير). هذا هو الحل الأفضل، لأنه بعد ذلك يمكنك التأكد في المستقبل أن العمل وفقا للاتفاقيات الحالية لتوحيد المقاييس ، والتي سوف عاجلا أو أجل يتم استبدال جميع القديم.
- أو أن تشمل هذا التعليق في السطر الأول أو الثاني:

```
# -*- coding:Latin-1 -*-
```

التعليق الذي فوق يخبر بيثون باستعمال السكريبت الخاص بك يشبه برموز ACSII المقابلة لأهم لغات أروبا الغربية (الفرنسية والألمانية والبرتغالية ...إلخ) المشفر على بايت واحد بعد ISO ..... وغالبا ما يشار إليه في التسمية بـ Latin-1 .

يمكن لبيثون التعامل على النحو المناسب مع الأحرف المشفرة للمعايير. لذلك لمساعدة بيثون يجب أن تضع التعليق في البرنامج النصي. من دون هذه الإشارة ، بيثون يفترض أنه تم ترميزه بـ utf-8[8]، تحت معيار اليونيكود الجديدة، والتي تم تطويرها لتوحيد التمثيل الرقمي لجميع الرموز من لغات العالم المختلفة بالإضافة لرموز الرياضيات والعلوم ...إلخ. هنالك العديد من الترميزات في هذا المعيار، سنعمل على هذه المسألة ، في الوقت الراهن، نعرف أن تماما أن الترميز الأكثر شيوعا في أجهزة الحاسوب الحديثة هو utf-8. في هذا النظام ، الترميز القياسي (ACSII) في بايت واحد، والذي يوفر بعض التوافق مع ترميز Latin-1 ولكن الأحرف الأخرى (بما في ذلك الأحرف المعلمة)يمكن ترميزها على 2 أو 3 أو حتى 4 بايت .

سوف نتعلم كيفية إدارة وتحويل هذه الترميزات ، عندما ندرسها بالتفصيل في معالجة الملفات النصية (الفصل التاسع)

تمارين

5.4 اكتب برنامجا يقوم بحساب متوازي المستطيلات اعتمادا على الطول والعرض والارتفاع

6.4 اكتب برنامجا يحول الثواني إلى سنوات وأشهر وأيام وساعات ودقائق وثواني (استخدم المعامل موديلو %).

7.4 اكتب برنامجا الذي يعرض أول 20 نتيجة لجدول الضرب على سبعة،مشيرا بنجمة مضاعفات العدد 3 .

على سبيل المثال : 7 14 21 * 28 35 42 * 49 ...

---

8 في الإصدارات السابقة لبيثون، كان الترميز الافتراضي هو ASCII. ولذلك كان يجب دائما وضع الترميز في بداية السكريبت (بما في ذلك Utf-8) .

8.4 اكتب برنامجا لحساب أول 50 نتيجة بجدول ضرب 13 ، لكن لن يظهر سوى التي تتقسم على 7 .

9.4 اكتب برنامجا يعرض التالي :

*

**

***

****

*****

******

****

5

أهم أنواع البيانات

في الفصل الثاني تعاملنا بالفعل مع العديد من أنواع البيانات : الأعداد الصحيحة أو الحقيقية والسلاسل النصية. حان الوقت الآن لنختبر ونقترب قليلا إلى هذه الأنواع من البيانات ، بالإضافة إلى التعرف على أنواع أخرى .

البيانات الرقمية

في تمارييننا حتى الآن ، لقد استخدمنا نوعين من البيانات : الصحيحة العادية والأعداد الحقيقية (و تسمى أيضا العائمة وأرقام النقطة الواحدة) سنسعى الآن جاهدين لتسليط الضوء على خصائص (و قيود) هذه المفاهيم .

العدد الصحيح

لنفترض أننا نريد عمل تعديل طفيف على تمارييننا السابقة بشأن تسلسل فيبوناتشي ،و ذلك للحصول على عرض أكبر عدد من المصطلحات .مبدئيا ، سنغير شرط الحلقة ، في السطر الثاني، مع `while c <50` : سنغيرها لنحصل على 49 شرط ، دعونا نغير التمرين قليلا ، لعرضه كنوع رئيسي للمتغير :

```
>>> a, b, c = 1, 1, 1
>>> while c <50:
        print(c, ":", b, type(b))
        a, b, c = b, a+b, c+1
...
...
...   (affichage des 43 premiers termes)
...
44 : 1134903170 <class 'int'>
45 : 1836311903 <class 'int'>
46 : 2971215073 <class 'int'>
47 : 4807526976 <class 'int'>
48 : 7778742049 <class 'int'>
49 : 12586269025 <class 'int'>
```

التمرين الذي قمنا به للتو ، يمكن للعدد الصحيح يمكننا من معرفة التمثيل الداخلي للأرقام في الحاسوب. لعلك تدرك في الواقع أن قلب الحاسوب يتكون من دوائر متكاملة إلكترونية (رقاقات السيليكون) تم دمجها بطريقة عالية ، ويمكنها أن تؤدي أكثر من مليار عملية في الثانية ، ولكن الأرقام الثنائية محدودة الحجم : حاليا[9]. أجهزة 32 بت ،8. مجموع القيم العشرية التي يمكن ترميزها في صيغة أرقام ثنائية 32 بت هو من 2147483648- إلى 2147483647+ .

العمليات على الأعداد الصحيحة بين هذين الحدين هي دائما سريعة جدا ، وذلك لأن المعالج قادر على التعامل معها بسرعة ، ومع ذلك عندما يتعلق الأمر مع أكبر الأعداد الصحيحة ، أو الأعداد الحقيقية (أرقام ذات فاصلة)، ستقوم المفسرات والمجمعات بعمل كبير بترميز\فك ترميز ، وهذا موجود في عمليات المعالج النهائية على الأعداد الثنائية ، 32 أقصى حد .

لا يوجد شيء يدعو للقلق حول هذه الاعتبارات التقنية. عندما تطلب من بيثون معالجة أي عدد صحيح فإنه يقوم بمعالجة هذه الأرقام وتحويلها إلى شكل من أشكال الأرقام الثنائية 32 بت كلما أمكنه ذلك، لتحسين سرعة الحساب وتوفير مساحة الذاكرة. وعندما تكون القيم التي يتم معالجتها أعدادا صحيحة تكون خارج الحدود المشار إليها أعلاه ، الترميز في الذاكرة الحاسوب يصبح أكثر تعقيدا، وبيثون يعالج هذا الرقم من قبل معالج يتتطلب عمليات متتالية عديدة. ويتم كل هذا تلقائيا ، دون الحاجة للقلق[10].

ستستطيع إذا مع بيثون حساب أعداد صحيحة تحتوي على أي أعداد كبيرة. ويقتصر هذا العدد في الواقع على حجم الذاكرة المتوفرة على الحاسوب المستخدم. إن الحسابات التي تتضمن أعدادًا كبيرة جدا

سيتم تقسيمها من قبل المترجم على عدة حسابات أكثر بساطة ، وهذا قد يتطلب قدرا أكبر من الوقت لمعالجتها في بعض الحالات .

على سبيل المثال :

```
>>> a, b, c = 3, 2, 1
>>> while c < 15:
        print(c, ": ", b)
        a, b, c = b, a*b, c+1

1 :  2
2 :  6
3 :  12
4 :  72
5 :  864
6 :  62208
7 :  53747712
```

[9] معظم أجهزة الحاسوب المكتبية تحتوي على معالج بسجلات 32 بت (حتى لو كان ثنائي النواة). سوف تكون معالجات 64 بت قريبا شائعة .

[10] في الإصدارات السابقة لبيثون لديها نوعان من الأعداد الصحيحة : `integer` و `long integer`، لكن التحويل بين هذين النوعين أصبح تلقائيا منذ الإصدار 2.2 .

```
8 :  3343537668096
9 :  179707499645975396352
10 :  600858794305667322270155425185792
11 :  107978831564966913814384922944738457859243070439030784
12 :  64880030544660752790736837369104977695001034284228042891827649456186234
582611607420928
13 :  70056698901118320029237641399576216921624545057972697917383692313271754
88362123506443467340026896520469610300883250624900843742470237847552
14 :  45452807645626579985636294048249351205168239870722946151401655655658398
64222761633581512382578246019698020614153674711609417355051422794795300591700
96950422693079038247634055829175296831946224503933501754776033004012758368256
>>>
```

في المثال أعلاه، قيم الأرقام تم عرضها بشكل سريع جدا ، لأن كل واحدة منها تساوي اثنين من الشروط السابقة. بالطبع، يمكنك الاستمرار في هذا التسلسل الرياضي إذا كنت تريد ، سيزداد نمو الأعداد الهائلة ، ولكن سرعة الحساب تنخفض تدريجيا .

> الأعداد الصحيحة للقيمة التي تشمل حدين مشار إليها في الأعلى تحتل كل واحدة 32 بت في ذاكرة الحاسوب. الأعداد الصحيحة الكبيرة جدا تحتل مكان متغير، إعتمادا على حجمها .

الأعداد الحقيقة

لقد تعرفنا في وقت سابق على هذا النوع من البيانات الرقمية ، عدد "نوع الحقيقي" مشار إليها في اللغة الإنكليزية من قبل أرقام النقطة ولذلك السبب يتم تسميتها أعدادا حقيقة في بيثون .

و هذا يسمح بالقيام بالعمليات الحسابية على أعداد كبيرة جدا أو أعداد صغيرة جدا (البيانات العلمية على سبيل المثال)، مع وجود درجة ثابتة من الدقة .

للحصول على أعداد رقمية بواسطة بيثون تعتبر من النوع العشري، يكفي أن يحتوي في صيغته على رمز مثل نقطة أو رقم أس (أو قوة) 10 .

على سبيل المثال :

```
        3.14       10.       .001      1e100      3.14e-10
```

يتم تفسير العبارات السابقة بأنها أرقام حقيقية تلقائيا من بيثون. دعونا إذا نجرب هذا النوع من البيانات في برنامج صغير (مستوحى من المثال أعلاه) :

```
>>> a, b, c = 1., 2., 1             # => " float - "عدد عشري" هي أعداد من نوع a وb
>>> while c <18:
...     a, b, c = b, b*a, c+1
...     print(b)

2.0
4.0
```

```
8.0
32.0
256.0
8192.0
2097152.0
17179869184.0
3.6028797019e+16
6.18970019643e+26
2.23007451985e+43
1.38034926936e+70
3.07828173409e+113
4.24910394253e+183
1.30799390526e+297
         Inf
         Inf
```

على الأرجح أنت فهمت ذلك جيدا ، وسنعرض لك مرة أخرى سلاسل من الأرقام ،التي تم إنهاؤها بسرعة كل واحدة منها تساوي نتيجتين سابقتين. في النتيجة التاسعة ، تحول بيثون تلقائيا إلى علمي ("E + رقم معناه "عشرة أضعاف أس\قوة الرقم) بعد 15، نحن نرى أنها فاقت الحد (بدون رسالة خطأ) ، الأرقام في الواقع كبيرة جدا ولاحظ ببساطة عبارة (Inf) لتعبر عن اللانهاية .

الأرقام الحقيقية المستخدمة في مثالنا تستخدم لمعالجة الأعداد (الموجبة والسالبة) تكون بين 10^{-308} و 10^{308} مع دقة. هذه الأرقام يتم ترميزها بطريقة معينة على 8 بايت (64 بت) في ذاكرة الحاسوب : جزء من الشيفرة يتوافق مع 12 رمز كبير وأخر بحجم (أس 10).

تمارين

1.5 اكتب برنامجا يحول زاوية بالرديان المقدمة أصلا بالدرجة والدقائق والثواني .

2.5 اكتب برنامجا يحول الدرجة والدقائق والثواني لزاوية زودت أصلا بالراديان .

3.5 اكتب برنامجا يحول درجة الحرارة من الدرجة المئوية إلى الفهرنهايت أو العكس
صيغة التحويل : $T_F = T_C \times 1,8 + 32$.

4.5 اكتب برنامجا يقوم بحساب الفؤائد المحققة سنويا لمدة 20 عاما ، برأس مال قدره 100 يورو وضعت في البنك بمعدل ربح ثابت بـ 4.8%

5.5 تقول أسطورة من الهند القديم أن لعبة الشطرنج اخترعت من قبل رجل يبلغ من العمر الحكمة، فشكره الملك وسيهديه أي هدية كمكافأة. أخبره العجوز أنه يريد ببساطة القليل من الأرز لأيامه السابقة. وعلى وجه التحديد ، عدد من حبات الأرز تكفي لوضعها في أنش جي واحدة على المربع الأول للعبة التي اخترعها ،إثنين للثانية و 4 للثالثة وهكذا إلى 64 مربع.
اكتب برنامج لحساب عدد حبات الأرز في المربعات ال 64 في اللعبة. بطريقتين:

- العدد الدقيق من الحبوب (عدد كامل)
- عدد الحبوب بالطريقة العلمية (العدد الحقيقي)

المعطيات الأبجدية

حتى الآن لم نتعامل سوى مع الأرقام، ولكن برنامج الحاسوب يحتاج أيضا إلى التعامل مع الحروف الأبجدية والكلمات والعبارات والسلاسل من الرموز. في معظم لغات البرمجة، وهذا من أجل استخدامها لأنواع خاصة من هياكل البيانات تسمى "السلاسل النصية" .

> سوف نتعلم في الفصل 10 أنه لا نبغي لنا أن نخلط بين مصطلحي "سلسلة نصية" و"سلسلة بايتات" كما فعلت لغات البرمجة القديمة من سوء استخدامهما (بما في ذلك الإصدارات القديمة لبيثون). وفي الوقت الراهن، يقوم بيثون بمعالجة متماسكة لكل السلاسل النصية، التي يمكن أن تكون جزءا من الحروف الأبجدية[11].

السلسلة

و يمكن تقريبا تعريف نوع البيانات السلسلة مثل أي سلسلة من الحروف. في سكريبت بيثون ، يمكن للمرء أن يعرف مثل هذه السلاسل من الأحرف. إما عن طريق علامات التنصيص المفردة أو علامات التنصيص الزوجية (علامة اقتباس). أمثلة على ذلك :

```
>>> phrase1 = 'les oeufs durs.'
>>> phrase2 = '"Oui", répondit-il,'
>>> phrase3 = "j'aime bien"
>>> print(phrase2, phrase3, phrase1)
"Oui", répondit-il, j'aime bien les oeufs durs.
```

المتغيرات الثلاثة **phrase2**، **phrase1** و **phrase3** متغيرات من نوع سلسلة .

لاحظ استخدام علامات الاقتباس لتحديد السلسلة التي توجد فيها علامة تنصيص مفردة، أو استخدم علامة الاقتباس المفردة ، لاحظ أيضا مرة أخرى أن دالة الطباعة تدرج مسافة بين العناصر المعروضة .

الرمز الخاص "\" (الخط المائل) يسمح ببعض المميزات الإضافية :

---

[11] و لذلك فإنها واحدة من المميزات الرئيسية للإصدار الجديد لبيثون (بيثون 3) مقارنة بالإصدارات السابقة. وفي هذه النسخة، البيانات من نوع string كانت سلسلة من البايتات وليس سلسلة من الحروف. وهذا لا يشكل مشكلة كبيرة في التعامل مع النصوص التي تحتوي فقط على الحروف الرئيسية للغات أوروبا الغربية، لأنه كان من الممكن ترميز كل هذه الأحرف في بايت واحد (على سبيل المثال، معيار Latin-1). وهذا أدى إلى صعوبة كبيرة إذا أردنا جميع الأحرف في نص واحد من الحروف الأبجدية المختلفة، أو ببساطة استخدام الحروف الأبجدية التي تحتوي على أكثر من 256 من الحروف والرموز الرياضية الخاصة ...إلخ. يمكنك العثور على المزيد من المعلومات حول هذا الموضوع في الفصل 10 .

- أولا، لأنها تتيح لك كتابة أسطر متعددة التي من شأنها أن تأخذ وقتا طويلا لاحتواها على سطر واحد(هذا ينطبق على أي نوع من التعليمات)
- ضمن السلسة يتم استخدامها لإدخال عدد من الرموز الخاصة(سطر جديد، علامة تنصيص مفردة، علامات الاقتباس، إلخ ...) أمثلة على ذلك :

```
>>> txt3 = '"N\'est-ce pas ?" répondit-elle.'
>>> print(txt3)
"N'est-ce pas ?" répondit-elle.
>>> Salut = "Ceci est une chaîne plutôt longue\n contenant plusieurs lignes \
... de texte (Ceci fonctionne\n de la même façon en C/C++.\n\
...     Notez que les blancs en début\n de ligne sont significatifs.\n"
>>> print(Salut)
Ceci est une chaîne plutôt longue
 contenant plusieurs lignes de texte (Ceci fonctionne
 de la même façon en C/C++.
    Notez que les blancs en début
 de ligne sont significatifs.
```

ملاحظات

- إن الرمز \n في السلسة معناه القفز إلى سطر جديد.
- إن الرمز \' لإدراج علامة تنصيص مفردة في سلسلة محددة بواسطة علامة تنصيص مفردة ونفس الشيء مع \" لإدخال علامات الاقتباس في سلسلة محددة بواسطة علامات الاقتباس .
- تذكر قواعد أسماء المتغيرات (يجب أن نحدد بدقة حالة الأحرف كبيرة أو صغيرة)

الاقتباس الثلاثي

لإدراج أحرف خاصة أو غريبة بسهولة في السلسلة دون استخدام رمز الخط المائل أو باستعمال رمز المائل نفسه في السلسلة ، يمكن للمرء استعمال سلسلة بعلامة اقتباس ثلاثية أو علامات التنصيص المفردة الثلاثية :

```
>>> a1 = """
...   Exemple de texte préformaté, c'est-à-dire
...      dont les indentations et les
...         caractères spéciaux \ ' " sont
... conservés sans
...       autre forme de procès."""
>>> print(a1)

  Exemple de texte préformaté, c'est-à-dire
     dont les indentations et les
        caractères spéciaux \ ' " sont
conservés sans
      autre forme de procès.
>>>
```

الوصول إلى الأحرف الفردية في السلسلة

السلاسل تمثل حالة خاصة من أنواع البيانات الأكثر عمومية تدعى مُركب. المركب المعين هو الذي يجمع في واحدة مجموعة منه أكثر بساطة في حالة وجود سلسلة ، على سبيل المثال، وهذه من الواضح أنه أبسط من الحروف نفسها. تبعا للظروف نحن نرغب بمعالجة السلسلة، وأحيانا الكائن الواحد ، وأحيانا مجموعة أحرف. لغة البرمجة بيثون تسمح بالوصول بشكل منفصل إلى كل من الأحرف في السلسة، كما سترى، وهذه ليست معقدة للغاية .

بيثون تفترض أن السلسلة هي كائن من فئة السلاسل، وهي التي طلبت مجموعات من العناصر. وهذا معناه ببساطة أن أحرف السلسلة ترتب دائما في ترتيب معين. ولذلك ، يمكن لكل حرف في السلسلة أن يكون له تسمية في مكانه في السلسلة، وذلك باستخدام المؤشر .

للوصول إلى حرف محدد، يجب علينا وضع اسم المتغير الذي يحتوي على السلسلة ونضع رقم المؤشر (و هو رقم موضع الحرف في السلسلة) داخل معقوفين .

انتبه : سوف يكون لديك فرصة للتحقق (لاحقا) من أن الأعداد المرقمة تبدأ دائما من الصفر (و ليس واحد). وهذا هو الحال بالنسبة لحروف السلسلة .

على سبيل المثال :

```
>>> ch = "Christine"
>>> print(ch[0], ch[3], ch[5])
C i t
```

تستطيع إعادة التمرين في الأعلى ، وهذه المرة استخدم حرفا أو حرفين غير أكسي non-ASCII. على عكس ما قد يحدث في بعض الحالات مع إصدارات بيثون قبل الإصدار 3.0، تحصل هنالك مفاجئة في النتيجة المتوقعة :

```
>>> ch ="Noël en Décembre"
>>> print(ch[1],ch[2],ch[3],ch[4],ch[8],ch[9],ch[10],ch[11],ch[12])
o ë l   D é c e m
```

لا داعي للقلق في الوقت الحالي حول كيفية قيام بيثون بالتخزين والتعامل مع الأحرف في ذاكرة الحاسوب. فقط اعلم أن هذه التقنية تستغل المعايير الدولية يونيكود. فيستطيع تمييز أي حرف من الحروف الأبجدية. لذلك يمكنك في نفس السلسلة خلط اللاتينية واليونانية والسيريلية والعربية ..إلخ والرموز الرياضية و إلخ …

> سوف نرى في الفصل 10 (انظر للصفحة 129) كيفية عرض الأحرف غير التي يمكن الوصول إليها مباشرة من لوحة المفاتيح .

العمليات الأساسية على السلاسل

بيثون يحتوي على العديد من الدالات التي تؤدي إلى تعاملات مختلفة على سلاسل (الأحرف الكبيرة \ الصغيرة، قطع أجزاء من السلسلة والبحث عن كلمات ... إلخ). مرة أخرى يجب أن تصبروا لأنه سيتم وضع شرح هذه الدالات في الفصل 10 (انظر الصفحة 129).

الآن ، يمكننا أن نعرف ببساطة أنه يمكن الوصول إلى كل حرف في السلسلة، كما هو موضح في الفقرة السابقة ، دعونا نضيف قليلا على ما سبق :

- نجمع العديد من السلاسل الصغيرة لبناء واحدة كبيرة. هذا ما يسمى التسلسل وهذا يتحقق في بيثون باستخدام الرمز + (هذا الرمز معناه إضافة سلسلة لسلسلة أخرى كما في الرياضيات وهذا يعمل في السلسلة النصية) مثال :

```
a = 'Petit poisson'
b = ' deviendra grand'
c = a + b
print(c)
petit poisson deviendra grand
```

- تحديد طول السلسة (أي عدد الأحرف) وذلك باستخدام **len**() :

```
>>> ch ='Georges'
>>> print(len(ch))
7
```

هذه الدالة تعمل بشكل جيد حتى لو كانت السلسلة تحتوي على أحرف أخرى :

```
>>> ch ='René'
>>> print(len(ch))
4
```

- تحويل رقم يمثل سلسلة نصية إلى عدد رقمي ، على سبيل المثال :

```
>>> ch = '8647'
>>> print(ch + 45)
→ *** خطأ *** : لا يمكن إضافة سلسلة إلى رقم
>>> n = int(ch)
>>> print(n + 65)
8712                    # نعم : يمكننا إضافة رقم إلى رقم آخر
```

في هذا المثال، دالة **int**() تحول السلسلة إلى عدد صحيح. سيكون ذلك من الممكن أيضا تحويل سلسلة أحرف إلى عدد حقيقي، وذلك باستخدام **float**().

تمارين

6.5 اكتب سكريبت يحدد إذا كانت السلسلة تحتوي على حرف "a" أو لا .

7.5 اكتب سكريبت يحسب عدد تواجد الحرف "a" في السلسلة .

8.5 اكتب سكريبت يقوم بنسخ سلسلة (في متغير جديد) وإدراج نجمة بين الأحرف.
على سبيل المثال ، "gaston" تصبح "g*a*s*t*o*n"

9.5 اكتب سكريبت يقوم بنسخ السلسلة (في متغير جديد) في إتجاه المعاكس.
على سبيا المثال : "Hisham" تصبح "mahsih".

10.5 استنادا إلى التمارين السابقة ، اكتب سكريبت يحدد إذا كانت السلسلة تعطي سياق متناظر أو لا (أي أن سلسلة يمكن قراءتها من الاتجاهين) ، مثلا "Radar" أو "SOS" .

القوائم (النهج الأول)

قدمت السلاسل التي ناقشناها في الجزء السابق مثال أولي من البيانات المركبة. هياكل البيانات التي تستخدم لتجميع مجموعة من القيم. وسوف تتعلم تدريجيا طريقة استخدام غيرها من مركبات عدة أنواع من البيانات. بما في ذلك ، القوائم والقواميس والمصفوفة المغلقة[12]. وسوف نناقش هنا أول هذه الأنواع الثلاثة، وهذا وهذا مختصر إلى حد ما. لكنه بالفعل موضوع واسع جدا، وسنعود إليه مرارا وتكرارا .

في بيثون، يمكننا تحديد قائمة من العناصر مفصولة بفواصل موضوعة كلها داخل نصفي مربع ، على سبيل المثال :

```
>>> jour = ['lundi', 'mardi', 'mercredi', 1800, 20.357, 'jeudi', 'vendredi']
>>> print(jour)
['lundi', 'mardi', 'mercredi', 1800, 20.357, 'jeudi', 'vendredi']
```

في هذا المثال، المتغير **jour** هو قائمة.

كما يمكن أن نرى في نفس المثال، فإن العناصر الفردية التي تشكل القائمة قد تكون من أنواع مختلفة. في هذا المثال، في الواقع، أول ثلاثة عناصر من السلسلة هي حروف والعنصر الرابع هو عدد صحيح والخامس هو عدد حقيقي وما إلى ذلك. (سنناقشها لاحقا ، يمكن للقائمة أن يكون أحد عناصرها قائمة !) في هذا الشأن ، القائمة قد تكون ""مصفوفة" (array) أو " متغير إنديسا" حسب لغة البرمجة .

لاحظ أيضا ، انه كما في السلاسل ، فإن القوائم هي سلسلة وهذا يعني أنها مرتبة. مختلف العناصر التي تشكل القائمة هي في الواقع دائما أعدت في نفس الترتيب. ويمكن الوصول إلى كل واحدة منها على حدة إذا كنا نعرف مؤشرها في القائمة. كما كان حال الأحرف في السلسلة ، يجب أن نعرف أن الترقيم يبدأ من الصفر وليس واحد .

أمثلة :

```
>>> jour = ['lundi', 'mardi', 'mercredi', 1800, 20.357, 'jeudi', 'vendredi']
>>> print(jour[2])
mercredi
```

12 يمكنك إنشاء أنواع من البيانات المركبة بنفسك. عندما تتحكم في مفهوم الصنف (انظر إلى صفحة 175).

```
>>> print(jour[4])
20.357
```

على عكس السلاسل ، والتي هي نوع من البيانات غير قابلة للتعديل (سيكون لدينا العديد من الفرص للعودة إليها مرة أخرى) ، فإنه من الممكن تغيير العناصر الفردية للقائمة :

```
>>> print(jour)
['lundi', 'mardi', 'mercredi', 1800, 20.357, 'jeudi', 'vendredi']
>>> jour[3] = jour[3] +47
>>> print(jour)
['lundi', 'mardi', 'mercredi', 1847, 20.357, 'jeudi', 'vendredi']
```

يمكننا استبدال بعض عناصر القائمة بأخرى ، كما هو مبين أدناه :

```
>>> jour[3] = 'Juillet'
>>> print(jour)
['lundi', 'mardi', 'mercredi', 'Juillet', 20.357, 'jeudi', 'vendredi']
```

دالة **len()**، التي استعملناها بالفعل في السلاسل ، ينطبق نفس مفهومها على القوائم ، لكن تقوم بإظهار عدد العناصر الموجودة في القائمة :

```
>>> print(len(jour))
7
```

دالة أخرى تقوم بحذف عنصر من القائمة (باستخدام المؤشر). وهذه الدالة هي **del()** [13] :

```
>>> del(jour[4])
>>> print(jour)
['lundi', 'mardi', 'mercredi', 'juillet', 'jeudi', 'vendredi']
```

و من الممكن إضافة عنصر إلى القائمة، ولكن للقيام بذلك، يجب علينا أن نعتبر القائمة كائن، وسوف نستخدم إحدى الطرق، سيتم شرح مفاهيم الحاسوب للكائن والأساليب في وقت لاحق ، ولكن ما يهمنا الآن "كيف تعمل" في القائمة :

```
>>> jour.append('samedi')
>>> print(jour)
['lundi', 'mardi', 'mercredi', 'juillet', 'jeudi', 'vendredi', 'samedi']
>>>
```

في السطر الأول من المثال أعلاه ، قمنا بطريقة **append()** لإضافة السبت للقائمة **jour**. لمن لا يعرف كلمة "append" تعني إضافة في الإنكليزية. ونحن نستطيع أن نفهم أن **append()** هو أسلوب يتم بطريقة أو بأخرى دمج أو إضافة عنصر إلى القائمة. البرامتر الذي يستخدم مع هذه الدالة هو بالطبع العنصر الذي نريد إضافته إلى نهاية القائمة .

---

13 في الواقع يوجد عدد متنوع من التقنيات التي تسمح لك بقطع قائمة إلى شرائح, وادراج مجموعات من العناصر, أو إزالة مجموعات أخرى ...إلخ. وذلك باستخدام تكوين جمل خاص يتضمن المؤشر .
و تسمى هذه المجموعة من التقنيات (و التي يمكن أيضا تطبيقها على السلاسل) بالتشريح. بوضع عدة مؤشرات بدلا من الواحد بين قوسين (نصف مربع) ثم نضيف اسم المتغير ؛ مثل jour[1:3] الذي هو ['mardi', 'mercredi'] .
و سيتم شرح بالتفصيل هذه التقنيات لاحقا (انظر لصفحة 129 وما يليها).

سوف نرى لاحقا مجموعة كبيرة من هذه الطرق (و هذا معناه دالات بنيت، أو بالأحرى "غلفت" في نوع قائمة). لاحظ أنه يتم تطبيق أسلوب الكائن من خلال ربطه مع النقطة. (الاسم الأول للمتغير الذي يشير للكائن ، ثم نقطة ثم اسم الأسلوب، وهذا يكون دائما برفقة زوج من الأقواس) .

كالسلاسل، سيتم التعمق في القوائم في وقت لاحق (انظر الصفحة 150). نحن لا نعرف ما يكفي للبدء في استخدام برنامجنا. يرجى قراءة المثال. لتحليل السكريبت الصغير أدناه والتعليق على كيفية عمل ذلك :

```
jour = ['dimanche','lundi','mardi','mercredi','jeudi','vendredi','samedi']
a, b = 0, 0
while a<25:
     a = a + 1
     b = a % 7
     print(a, jour[b])
```

السطر الخامس في هذا المثال يستخدم العامل "موديلو" الذي درسناه في وقت سابق ويمكن أن يكون ذا فائدة كبيرة في البرمجة. ويتم تمثيله بـ % في العديد من لغات البرمجة (بما في ذلك بيثون). حسنا ، ما هي العملية التي يقوم بها هذا العامل ؟

تمارين

11.5 انظر في القوائم التالية :

```
t1 = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
t2 = ['Janvier', 'Février', 'Mars', 'Avril', 'Mai', 'Juin',
      'Juillet', 'Août', 'Septembre', 'Octobre', 'Novembre', 'Décembre']
```

اكتب برنامجا صغيرا يقوم بإنشاء قائمة جديدة **t3**. وهذه القائمة يجب أن تحتوي على جميع العناصر من القائمتين بالتناوب ، بحيث يتبع كل اسم عدد أيام الشهر .

```
['Janvier',31,'Février',28,'Mars',31, etc...].
```

12.5 اكتب برنامجا يعرض بشكل صحيح جميع عناصر القائمة. إذا طبقت على سبيل المثال إلى **t2** الأوامر المذكورة أعلاه يجب أن تحصل على :

```
Janvier Février Mars Avril Mai Juin Juillet Août Septembre Octobre Novembre
Décembre
```

13.5 اكتب البرنامج الذي يبحث في القائمة على أكبر عنصر. فمثلا إذا طبقته على قائمة

```
[32, 5, 12, 8, 3, 75, 2, 15]
```

يجب عليك أن تحصل على :

```
le plus grand élément de cette liste a la valeur 75.
```

14.5 اكتب برنامجا الذي يقوم بفحص كل عناصر واحدا تلو الآخر لقائمة أرقام (على سبيل المثال من التمرين السابق) لإنشاء قائمتين جديدتين. تحتوي الأولى على الأرقام الزوجية والثانية على الأرقام الفردية ، نصيحة : استخدم المعامل موديلو المذكور أعلاه .

15.5 اكتب برنامج الذي يحلل واحد تلو الأخر كل عناصر القائمة من الكلمات، على سبيل المثال:

```
['Jean', 'Maximilien', 'Brigitte', 'Sonia', 'Jean-Pierre', 'Sandra']
```

لتولد قائمتين جديدتين، واحدة للكلمات أقل من 6 أحرف، وواحدة أخرى 6 أحرف أو أكثر.

6

الدالات المعرفة مسبقا

واحدة من أهم مفاهيم في البرمجة هي الدالة[14]. الدالة تجعلك تحلل برنامجا معقدا إلى سلسلة من البرامج البسيطة، والتي بدورها يمكن تقسيمها إلى أجزاء أصغر، وهكذا. وإضافة إلى ذلك فإن الدالة قابلة لإعادة الاستخدام فمثلا إذا كان لدينا دالة تحسب الجذر التربيعي، يمكن أن نستخدمها في كل مكان في برنامجنا دون الحاجة إلى إعادة كتابتها كل مرة .

الدالة print()

لقد تعرفنا سابقا بهذه الدالة. أردت أن أشير هنا إلى أنه يسمح بعرض أي عدد من القيم المتوفرة ك برامترات (و هذا يعني ما بين أقواس). افتراضيا، سيتم فصل هذه القيم عن بعضها البعض بمسافة، وفي النهاية يتم القفز إلى سطر جديد .

يمكننا استبدال المسافة الافتراضية بأي شيء آخر (أو حتى بلا شيء) من خلال البارامتر **sep**. مثال ::

```
>>> print("Bonjour", "à", "tous", sep ="*")
Bonjour*à*tous
 >>> print("Bonjour", "à", "tous", sep ="")
Bonjouràtous
```

و يمكنك استبدال القفز إلى سطر جديد باستخدام البارامتر **end** :

```
>>> n =0
>>> while n<6:
...     print("zut", end ="")
...     n = n+1
...
zutzutzutzutzutzut
```

---

14 في بيثون يستخدم مصطلح "دالة" للإشارة إلى الدالات الحقيقية ويستخدم أيضا للإشارة إلى الإجراءات. وسوف نشرح لاحقا التمييز بين هذين المفهومين المتشابهين .

التفاعل مع المستخدم : الدالة input()

حاليا معظم مدخلات المستخدم تتم عن طريق (إدخال برامترات، النقر بواسط الفأرة، الضغط على زر في لوحة المفاتيح، إلخ.). في سكريبت الوضع النصي (مثل التي صنعناها حتى الآن) ، أبسط طريقة هي استخدام الدالة **input()**. هذه الدالة تسبب توقف البرنامج لتدعو المستخدم لإدخال حروف من لوحة المفاتيح ويجب أن ينتهي مع الضغط على زر الإدخال (Enter). وعندما يضغط المستخدم زر الإدخال تقوم الدالة بأخذ ما كتبه المستخدم ويمكن إسناد قيمة لأي متغير بهذه الدالة أو تحويله .

يمكن للمبرمج استدعاء دالة **input()** ، وترك الأقواس فارغة ويمكنه أيضا أن يضع بارامتر به رسالة تفسيرية للمستخدم ، على سبيل المثال :

```
prenom = input("Entrez votre prénom : ")
print("Bonjour,", prenom)
```

أو :

```
print("Veuillez entrer un nombre positif quelconque : ", end=" ")
ch = input()
nn = int(ch)              # تحويل السلسلة إلى عدد صحيح
print("Le carré de", nn, "vaut", nn**2)
```

لاحظ أن دالة **input()** تقوم دائما بإرجاع سلسلة نصية[15]. فإذا كنت تريد أن يقوم المستخدم بإدخال قيمة رقمية، سوف تحتاج إلى تحويل قيمة المدخلات (و التي ستكون سلسلة نصية) إلى النوع الرقمي الذي يناسبك، من خلال وضع دالة **int()** (إذا كنت تتوقع عدد صحيح) أو **float()** (إذا كنت تتوقع عدد حقيقي). على سبيل المثال :

```
>>> a = input("Entrez une donnée numérique : ")
Entrez une donnée numérique : 52.37
>>> type(a)
<class 'str'>
>>> b = float(a)        # تحويل السلسلة إلى عدد حقيقي
>>> type(b)
<class 'float'>
```

استدعاء وحدة دالات

لقد تعرفت بالفعل على العديد من دالات اللغة ، مثلا دالة **len()**، الذي يسمح بمعرفة طول السلسلة. مع ذلك ، ليس من الممكن دمج جميع الدالات في بيثون القياسية ، لأنه يوجد عدد لامتناهي من الدالات : والتي سوف تتعلم قريبا كيفية صنع دالات جديدة بنفسك. الدالات في بيثون القياسية هي قليلة نسبيا : فيوجد بها فقط الدالات التي يتم استخدامها بشكل متكرر جدا ، والبعض الأخر يتم وضعها في ملفات خاصة تدعى وحدات .

---

15 في الإصدارات السابقة لبيثون. القيم التي يتم إرجاعها من خلال input() كانت من نوع متغير إعتمادا على ما قام المستخدم بإدخاله. السلوك الحالي كان سابقا دالة raw_input()، والذي يفضله معظم المبرمجين.

> الوحدات هي ملفات التي تحتوي على مجموعات من الدالات[16].

سترى في وقت لاحق أن تقسيم البرنامج إلى عدة ملفات أمر مريح لسهولة الصيانة. تطبيق بيثون يتألف من برنامج رئيسي يرافقه وحدة واحدة أو أكثر ، كل منها يحتوي على تعريفات عدد من الدالات الإضافية .

هنالك العديد من وحدات بيثون التي يتم توفيرها تلقائيا مع بيثون. تستطيع العثور على غيرها من مختلف المصادر. كثيرا ما نحاول جمع مجموعة من دالات لها ذات الصلة في نفس الوحدة، التي نسميها مكتبات .

وحدة math على سبيل المثال، تحتوي على تعريفات الدالات الرياضية مثل الجذر التربيعي ... إلخ. لاستخدام هذه المميزات، يمكنك ببساطة إدراج السطر التالي في بداية السكريبت الخاص بك :

```
from math import *
```

هذا السطر يخبر بيثون أن يتم إدراج جميع الدالات في البرنامج الحالي (هذا معناه أن الرمز * "الجوكر") لوحدة الرياضيات، والتي تحتوي على دالات رياضية مبرمجة مسبقا .

في داخل السكريبت ، سوف تكتب على سبيل المثال :

`racine = sqrt(nombre)` لتعيين المتغير `racine` الجذر التربيعي لـ `nombre`،

`sinusx = sin(angle)` لتعيين المتغير `sinusx` جيب `angle` (بالراديان!)، إلخ ....

على سبيل المثال :

```
# تجربة : استخدام دالات وحدة math

from math import *

nombre = 121
angle = pi/6          # إذا كانت 30°
                      # (مكتبة الرياضيات تتضمن أيضا تعريف pi)
print("racine carrée de", nombre, "=", sqrt(nombre))
print("sinus de", angle, "radians", "=", sin(angle))
```

عندما تشغل هذا السكريبت سوف يظهر التالي :

```
racine carrée de 121 = 11.0
sinus de 0.523598775598 radians = 0.5
```

هذا المثال القصير يوضح بشكل جيدة بعض الخصائص المهمة للدالات :

- يجب كتابة اسم الدالة بجانب قوسين مثال : **sqrt()**

[16] بالمعنى الدقيق للكلمة. يمكن للوحدة أن تحتوي أيضا على معرفات المتغيرات وكذلك الأصناف. وسوف نترك هذه التفاصيل جانبا لبعض الوقت)

- داخل القوسين يمكننا كتابة برامتر واحد أو أكثر مثال : **sqrt(121)**
- تقوم الدالة بإرجاع قيمة، مثال : **11.0**

سنضع كل هذا في الصفحات القادمة. يرجى ملاحظة أن الدالات المستخدمة هنا ليست سوى مثال أولي. فإن نظرت سريعا في وثائق مكتبات بيثون سوف تجد العديد من دالات لتنفيذ العديد من المهام ، بما في ذلك خوارزميات الرياضية المعقدة (وتستخدم على نطاق واسع في الجامعات التي تستخدم بيثون لحل المشاكل العلمية ذات المستوى العالي). ليس هنالك شك هنا لتوفير قائمة مفصلة. مثل هذه القائمة التي يمكن الوصول إليها بسهولة في النظام باستخدام بيثون :

Documentation HTML → Python documentation → Modules index → math

أي : وثائق HTML ← وثائق بيثون ← فهرست الوحدات ← math (رياضيات)

في الفصل القادم سوف نتعلم كيفية صنع دوال خاصة بنا .

تمارين

> في جميع التمارين استخدم الدالة ***input()*** لإدخال البيانات.

1.6 اكتب برنامجا لتحويل ميل/ساعة إلى المتر/ الثانية وإلى كم/ثانية (لا تنسى أن 1 ميل : 1609 متر) .

2.6 اكتب برنامجا يقوم بحساب محيط ومساحة أي مثلث. (سيدخل المستخدم أضلاع المثلث الثلاثة) .

(تذكير يتم حساب مساحة أي مثلث باستخدام هذه الصيغة :

$$S=\sqrt{d\cdot(d-a)\cdot(d-b)\cdot(d-c)}$$

حيث d هو طول نصف المحيط و c ، b ، a هي أطرافه الثلاثة)

3.6 اكتب برنامجا يقوم بحساب فترة من بندول بسيط لمدة معينة.

الصيغة لحساب فترة البندول البسيط هو : $T=2\pi\sqrt{\frac{l}{g}}$ ،

L تمثل قيمة البندول و G تمثل قيمة التسارع الناتج عن الجاذبية بدل الخبرة .

4.6 اكتب برنامجا يسمح لك بوضع القيم في القائمة. البرنامج يجب أن يحتوي على حلقة، ويتم طلب من المستخدم الحصول على القيم وعند إنتهائه يضغط على زر الإدخال دون أن يكتب شيئا ويتم إنهاء البرنامج. مع إنهاء يتم عرض القائمة، مثال على العملية :

```
Veuillez entrer une valeur : 25
Veuillez entrer une valeur : 18
Veuillez entrer une valeur : 6284
Veuillez entrer une valeur :
[25, 18, 6284]
```

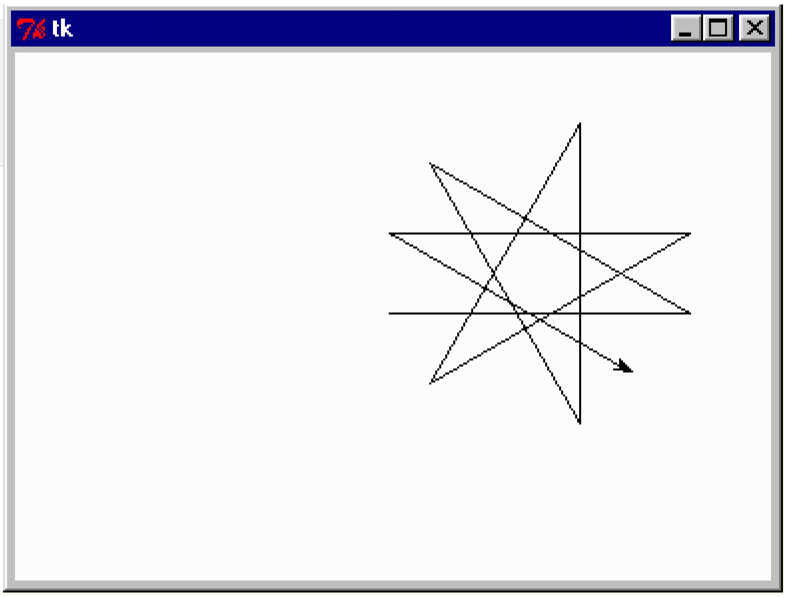
الاسترخاء قليلا مع وحدة turtle

كما رأينا سابقا، واحدة من أهم مميزات بيثون هو أنه من السهل للغاية إضافة العديد من الدالات عن طريق استيراد الوحدات .

لتوضيح هذا، وللحصول على بعض المتعة مع الكائنات الأخرى (بدل الأرقام) ، سوف نستكشف وحدة بيثون الـتي تسمى بـ " رسومات السلحفات " وهذا يعني، رسوم هندسية مناظرة إلى المسار التي خلفها "سلحفاة" صغيرة ظاهرية ، سنراقب تحركاتها على شاشة الحاسوب باستخدام تعليمات بسيطة .

بتفعيل هذه الساحافة فهي لعبة حقيقية للأطفال. بدلا من إعطاء تفسيرات طويلة ، نحن ندعوك لمحاولتها على الفور :

```
>>> from turtle import *
>>> forward(120)
>>> left(90)
>>> color('red')
>>> forward(80)
```



التمرين سيصبح أكثر غنى إذا تم استخدام الحلقات

```
>>> reset()
>>> a = 0
>>> while a <12:
        a = a +1
        forward(150)
        left(150)
```

تحذير : قبل أن تبدأ تشغيل هذا السكريبت، تأكد دائما من أنه لا يتضمن حلقة بدون إنهائها (راجع الصفحة 30)، لإنه إذا كان به حلقة غير منتهية قد تكون غير قادر على استعادة السيطرة على العمليات(و خاصة على نظام تشغيل ويندوز) .

استمتع بكتابة السكريبتات التي تجعل من الرسوم التالية علامة لتقدمك. والدالات الرئيسية المتاحة لك في وحدة turtle هي :

reset() إزالة كل شيء والبدء من جديد

goto(x, y) الذهاب إلى مكان الإحداثية x, y

forward(distance)	التقدم إلى الأمام لمسافة معينة
backward(distance)	الرجوع إلى الخلف
up()	(رفع القلم (المضي قدما دون الرسم
down()	(إخفض قلم الرصاص (لبدء الرسم
color(couleur)	لون القناة محدد مسبقا
left(angle)	(الاتجاه يسارا بزاوية معينة (بالدرجة
right(angle)	اتجه إلى اليمين
width(épaisseur)	حدد سمك الخط
fill(1)	تعبئة محيط مغلق باستخدام لون محدد
write(texte)	يجب على النص أن يكون سلسلة نصية

تعبير حقيقي\مزيف

عندما يحتوي البرنامج على عبارات معينة مثل **while** و **if**. يجب على الحاسوب الذي يشغل البرنامج فحص صحة الشرط ، فهو يتأكد إذا كان شرط التعبير صحيحا أو خاطئا. على سبيل المثال : الحلقة التي تبدأ بـ `while c<20:` ستبقى تعمل مادام c أصغر من 20 .

لكن كيف يمكن لجهاز الحاسوب تحديد ما إذا كان الشرط صحيحا أو خاطئا ؟

في الواقع أنت تعرف مسبقا أن الحاسوب يعالج الأرقام بدقة. يجب أولا على الحاسوب أن يحول المعلومات إلى قيم رقمية. وهذا ينطبق على مفهوم الصح /خطأ. في بيثون ، كما هو الحال في سي والعديد من لغات البرمجة الأخرى، يعتبر الحاسوب أي قيمة رقمية أخرى غير الصفر هي "صحيحة". فقط الصفر هو "الخطأ" ، على سبيل المثال :

```
ch = input('Entrez un nombre entier quelconque')
n =int(ch)
if n:
    print("vrai")
else:
    print("faux")
```

السكريبت الصغير في الأعلى سيظهر خطأ إذا أدخلت أي قيمة بخلاف الصفر ، وإذا أدخلت قيمة أخرى ستحصل على واحد .

الوسائل المذكورة أعلاه يجب أن يتم فحص التعبير، مثل `a > 5`، أولا سيحول الحاسوب هذه العبارة إلى قيمة رقمية (1 إذا كان التعبير صحيحا ، وصفر إذا كان التعبير خاطئا). هذا ليس واضحا كثيرا، وذلك لأن مفسر بيثون يترجم هاتين القيمتين إلى `True` أو `False`. على سبيل المثال :

```
>>> a, b = 3, 8
>>> c = (a < b)
>>> d = (a > b)
>>> c
True
>>> d
False
```

سيتم تخزين نتيجة التعبير a < b (صحيح) في متغير c. وبالمثل نتيجة لمتغير عكسي ، يتم التسجيل في المتغير d[17].

باستخدام القليل من الحيل ، يمكننا التحقق إذا كانت هذه القيم صحيحة أو خاطئة (هي في حقيقة الرقمين 1 و 0) .

```
>>> accord = ["non", "oui"]
>>> accord[d]
non
>>> accord[c]
oui
```

باستخدام المتغيرين c و d مؤشرات لاسترداد العناصر من قائمة accord، نحن نتأكد أن خطأ = 0 وصحيح = 1 .

السكريبت الصغير التالي يشبه كثيرا السكريبت السابق. فهو يسمح لنا باختبار الحرف صحيح أو خطأ لسلسلة نصية :

```
ch = input("Entrez une chaîne de caractères quelconque")
if ch:
    print("vrai")
else:
    print("faux")
```

سوف تحصل على "خطأ " لكل سلسلة فارغة ، و"صحيح" لكن سلسلة تحتوي على الأقل على حرف. فهل تستطيع أن تختبر بنفس الطريقة إذا كانت السلسلة فارغة تظهر "خطأ" وإذا كانت تحتوي على أي شيء تكون "صحيحة"[18].

العبارة : if ch:ـ في السطر الثالي من هذا المثال ،هو ما يعادل if ch !="" :من وجهة نظرنا كبشر ، أما بالنسبة للحاسوب فهي ليست كذلك ، فالعبارة if ch: هي للتحقق مباشر من أن قيمة المتغير ch هو متغير فارغ أو لا ، كما نراه نحن، اما في العبارة if ch != "": يتطلب البدء في مقارنة محتوى ch بقيمة المقدمة التي وضعناها في برنامجنا (سلسلة فارغة)، ثم اختبار نتيجة هذه المقارنة صحيحة أو خاطئة (أو بعبارة أخرى، يتأكد إذا كانت النتيجة صحيحة أو خاطئة). لذا فهو يتطلب عمليتين متتاليتين، العبارة الأولى هي الأكثر كفائة .

للأسباب نفسها، في هذا السكريبت :

```
ch =input("Veuillez entrer un nombre : ")
n =int(ch)
if n % 2:
```

[17] هذه المتغيرات من نوع صحيح خاص ؛ نوع "منطقي – Boolean". المتغيرات من هذا النوع لا يمكن أن تأخذ سوى قيمتين True و False (في الواقع، 1 و 0).

[18] هياكل البيانات الأخرى تتصرف بطريقة متشابهة. سوف تدرس المصفوفات المغلقة والقواميس في الفصل 10، والتي ستكون "خطأ" في حالة أنها فارغة، وصحيحة إذا كان لديها محتوى .

```
    print("Il s'agit d'un nombre impair.")
else:
    print("Il s'agit d'un nombre pair.")
```

و هو أكثر فاعلية ما فعلناه لبرمجة السطر الثالث ، كما فعلنا في الأعلى، أو بالأحرى كتابة هذا بشكل واضح `if n % 2 != 0` ، لأن هذه الصيغة تتطلب من جهاز الحاسوب أداء عمليتي مقارنة متتالية بدلا من واحدة .

هذا التعليل "قريب من الحاسوب"، ربما سيبدو لك خفيا في البداية، ولكن نعتقد أن هذا الشكل من الكتابة سوف تتعلمه بسرعة.

مراجعة

في ما يلي، لن نتعلم مفاهيم جديدة، ولكن مجرد استخدام كل ما تعلمناه سابقا لصنع برامج صغيرة حقيقية .

التحكم في تلقيم التنفيذ - باستخدام قائمة بسيطة

دعونا نبدأ مع عودة صغيرة لفروع الجمل الشرطية (ربما هذه مجموعة التعليمات الأكثر الأهمية في أي لغة):

```
# استخدام قائمة مع شروط متفرعة

print("Ce script recherche le plus grand de trois nombres")
print("Veuillez entrer trois nombres séparés par des virgules : ")
ch =input()
# ملاحظة : إن ربط الدالتين list() وeval() يسمح بتحويل
# في القائمة جميع سلاسل مفصولة بفواصل :19
nn = list(eval(ch))
max, index = nn[0], 'premier'
if nn[1] > max:                    # لا تنسى النقطتين !
    max = nn[1]
    index = 'second'
if nn[2] > max:
    max = nn[2]
    index = 'troisième'
print("Le plus grand de ces nombres est", max)
print("Ce nombre est le", index, "de votre liste.")
```

في هذا التمرين، نجد مرة أخرى مفهوم "تعليمة الكتلة" ، التي بدأنها بالفعل في الفصلين 3 و 4 ، والتي كان يجب عليك استيعابها (للتذكير، كتل التعليمات محددة بمسافة). بعد أول عبارة if، على سبيل المثال ، هنالك نوعان من بادئة أسطر تحديد كتلة البيانات. سيتم تنفيذ هذه البيانات إذا كنا الشرط `nn[1] > max` صحيح.

19 في الحقيقة، إن دالة `eval()` تقوم بفحص محتوى السلسلة التي تم توفيرها على شكل برامتر كتعبير بيثون الذي يجب أن يتم إرجاع نتيجته، على سبيل المثال: `eval("7 + 5")` يجب أن يتم إرجاع العدد الصحيح 12. فإذا قمت بتوفير سلسلة من القيم مفصولة بفواصل، وهذه تتوافق مع المصفوفة المغلقة. المصفوفات المغلقة هي متسلسلات متصلة بقوائم. وسيتم شرحها في الفصل العاشر (أنظر إلى صفحة 163).

السطر التالي (استعمال عبارة **if**) ثانية). لم يبدأ ببادئة ، ولذلك فإن هذا السطر يتم تعريفه كجزء من جسم البرنامج. فيتم تنفيذ هذا الجزء دائما ، في حين أن السطرين التاليين (و التي يتم تنفيذها حتى الآن ككتلة) لن يتم تنفيذه إلا إذا كان الشرط `nn[2] > max` صحيحا.

نتقدم بنفس المنطق، ونرى أن على السطرين الأخيرين هما جزء من الكتلة الرئيسية والتي يتم تنفيذها دائما .

حلقة while – التداخل

نواصل السير في نفس المسار عن طريق دمج هياكل أخرى :

```
1#  #  تعليمة مركبة <while> - <if> - <elif> - <else>
2#
3#  print('Choisissez un nombre de 1 à 3 (ou 0 pour terminer) ', end=' ')
4#  ch = input()
5#  a = int(ch)             # تحويل السلسلة المدخلة إلى عدد صحيح
6#  while a:                # تعادل à : < while a != 0: >
7#      if a == 1:
8#          print("Vous avez choisi un :")
9#          print("le premier, l'unique, l'unité ..."
10#     elif a == 2:
11#         print("Vous préférez le deux :")
12#         print("la paire, le couple, le duo ...")
13#     elif a == 3:
14#         print("Vous optez pour le plus grand des trois :")
15#         print("le trio, la trinité, le triplet ...")
16#     else :
17#         print("Un nombre entre UN et TROIS, s.v.p.")
18#     print('Choisissez un nombre de 1 à 3 (ou 0 pour terminer) ', end =' ')
19#     a = int(input())
20# print("Vous avez entré zéro :")
21# print("L'exercice est donc terminé.")
```

نجد هنا أن حلقة **while** مرتبطة لمجموعة عبارة **elif**، **if** و **else**. لاحظ مرة أخرى بنية المنطقية للبرنامج المصنوع بمساعدة التبويتات (... لا تنسى الرمز ":" في نهاية السطر)

في السطر السادس، يتم استخدام عبارة **while** كما هو موضح في الصفحة 57 : لتفهمها فقط تذكر أن كل القيم الرقمية غير الصفر تعتبر صحيحة من قبل مفسر بيثون. تستطيع تغيير هذا الشكل من الكتابة بـ `while a != 0` : إذا كنت تفضل هذا (لنتشكر هنا أن المعامل المقارنة != "تختلف عن ")، لكن أقل فاعلية .

هذه "حلقة while" تحفز باستجواب بعد كل إجابة من المستخدم (على الأقل حتى قرر الخروج ولم يدخل أي قيمة)

في داخل الحلقة، نجد مجموعة من العبارات **elif**، **if** و **else** (من السطر 7 إلى السطر 17)، التي توجه تدفق البرنامج إلى أماكن مختلفة ، ثم تعليمة **print()** والتعليمة **input()** (السطر 18 و 19) يتم تشغيله في جميع الحالات : يرجى ملاحظة مستوى مسافة البادئة، والذي هو نفس كتلة **elif**، **if** و **else**. بعد هذه التعليمات، تستأنف حلقة البرنامج تنفيذها مع العبارة **while** (السطر 6).

في السطر 19، استخدامنا تركيبة الكتابة لكتابة الكود أكثر إيجازا، وهو ما يعادل السطرين 4 و 5 مجتمعين .

يتم تنفيذ عبارتي **print()** الأخيرتين (السطر 20 و 21) اللتين ستعملان عند الخروج من الحلقة .

تمارين

5.6 ما يفعل هذا البرنامج (في الأسفل)، في هذه الحالات الأربعة : إذا كان **a** يساوي 1، 2، 3 أو 15 ؟

```
if a !=2:
    print('perdu')
elif a ==3:
    print('un instant, s.v.p.')
else :
    print('gagné')
```

6.6 ماذا تفعل هذه البرامج ؟

```
a)  a = 5
    b = 2
    if (a==5) & (b<2):
        print('"&" signifie "et"; on peut aussi utiliser\
              le mot "and"')
b)  a, b = 2, 4
    if (a==4) or (b!=4):
        print('gagné')
    elif (a==4) or (b==4):
        print('presque gagné')
c)  a = 1
    if not a:
        print('gagné')
    elif a:
        print('perdu')
```

7.6 جرب البرنامج c) مع a = 0 بدلا من a =1. ماذا حدث ؟ هل دخل !

8.6 اكتب برنامجا يقوم بتحديد عددين صحيحين a و b، ثم يضيف رقمي الضرب لـ 3 و 5 بين هذين الحدين. على سبيل المثال a = 0 و b = 32 ويجب أن يكون الناتج 0 + 15 + 30 = 45.

قم بتعديل طفيف على البرنامج لإضافة أرقام ضرب 3 أو 5 بين حدي a و b. وبحدي 0 و 32 يكون الناتج : 0 + 3 + 5 + 6 + 9 + 10 + 12 + 15 + 18 + 20 + 21 + 24 + 25 + 27 + 30 = 225.

9.6 اكتب برنامجا يحدد إذا كانت السنة كبيسة أو لا ، والسنة الكبيسة هي السنة التي تقبل القسمة على 4. ولن تكون كبيسة إذا كان A من مضعفات 100 (على الأقل A ليس من مضعفات رقم 400) .

10.6 اكتب البرنامج الذي يطلب من المستخدم اسمه وجنسه (ذكر أو أنثى) وبناءا على هذه البيانات، البرنامج سيعرض " عزيزي السيد " أو عزيزتي السيدة " متبوعة باسم الشخص .

11.6 أطلب من المستخدم إدخال 3 أطوال (a و b و c). حدد ما إذا كان من الممكن إنشاء مثلث بمساعدة هذه الأطوال الثلاثة. ثم حدد ما إذا كان هذا المثلث قائمة الزاوية أو متساوي الضلعين أو متساوي الأضلاع أو إلخ. انتبه : يمكن أن يكون المثلث متساوي الضلعين .

12.6 اكتب برنامجا يطلب من المستخدم إدخال رقم : ثم يعرض له الجذر التربيعي لهذا العدد ، فإذا لم يوجد جذر تربيعي لذلك الرقم تظهر له رسالة تخبره بذلك .

13.6 اكتب برنامجا يحول علامة المدرسة التي أدخلها المستخدم بشكل نقاط (على سبيل المثال 27 من 85) ،إلى درجة قياسية مثل التالي :

```
Note                 Appréciation
N >= 80 %                  A
80 % > N >= 60 %           B
60 % > N >= 50 %           C
50 % > N >= 40 %           D
N < 40 %                   E
```

14.6 انظر في القائمة التالية :

```
['Jean-Michel', 'Marc', 'Vanessa', 'Anne', 'Maximilien',
 'Alexandre-Benoît', 'Louise']
```

اكتب برنامجا يعرض كل هذه الأسماء مع عدد الحروف التي تتكون منها

15.6 اكتب حلقة برنامج تطلب من المستخدم إدخال نتائج الطلاب. الحلقة لا تتوقف إلا عندما يدخل المستخدم قيمة سالبة. مع النتائج التي تم إدخالها ، يتم وضعها في قائمة. بعد كل دخول نتيجة (بالتالي كل تكرار للحلقة)، يظهر البرنامج عدد النتائج التي تم إدخالها، الدرجة الأكثر تقديرا والدرجة الأقل ، ومعدل جميع النتائج .

16.6 اكتب سكريبت يظهر قيمة قوة الجاذبية التي تعمل بين كتلتين 10 000 kg ، لمسافة تزيد هندسيا عن 2 ، بداية من 5 سم (0,05 متر) .

تخضع قوة الجاذبية لهذه الصيغة : $F=6,67.10^{-11}\cdot\frac{m\cdot m'}{d^2}$

مثال :

```
d = .05 m :  la force vaut  2.668 N
d = .1 m  :  la force vaut  0.667 N
d = .2 m  :  la force vaut  0.167 N
d = .4 m  :  la force vaut  0.0417 N
```

إلخ.

7

الدالات الأصلية

البرمجة هي فن تعليم الحاسوب لأداء مهام لم يكن قادرا على أدائها سابقا. أحد الأساليب الأكثر إثارة للإهتمام لتحقيق هذا هو إضافة تعليمات جديدة للغة البرمجة التي تستخدمها، في شكل دالة أصلية .

تعريف الدالة

السكريبتات التي كتبتها حتى الآن قصيرة للغاية ، وذلك لأن هدفها هو تعلم أساسيات اللغة ، بمجرد أن تبدأ بتطوير مشاريع حقيقية، سوف تواجه الكثير من المشاكل المعقدة، وتبدأ أسطر البرنامج بالتراكم ...

الطريقة الفعالة لحل الكثير من المشاكل هي تقسيم المشاكل إلى مجموعة مشاكل صغيرة أكثر بساطة لدراسة كل واحدة على حدة (يمكن لهذه المشاكل الصغيرة أن تحلل نفسها بنفسها بدورها) ومن المهم أن يتم تقسيم هذه المشاكل بشكل صحيح في خوارزميات[20] ويجب أن تكون واضحة .

و من ناحية أخرى، فإنه غالبا ما تستخدم نفس تسلسل التعليمات مرارا وتكرارا في أحد البرامج، ولن يكون جيدا إعادة كتابة الكود كل مرة .

الدالات[21] والكائنات هي هياكل مختلفة من الوظائف الفرعية التي تم تخيلها من قبل المبرمجين للغات عالية المستوى لحل الصعوبات المذكورة أعلاه. سنقوم هنا للمرة الأولى بشرح تعريف الدالات في بيثون. وسوف نناقش الكائنات والصفوف في وقت لاحق .

لقد التقيت بالفعل مع الدالات المبرمجة سابقا لأداء مهام مختلفة. سوف نتعلم الآن كيف صنع دالات بأنفسنا .

20 الخوارزمية هي سلسلة مفصلة من العمليات المطلوبة من أجل حل مشكلة.

21 و يوجد في لغات البرمجة الأخرى روتينات (و التي تسمى أيضا برامج فرعية) والإجراءات. لا يوجد روتينات في بيثون. وبالمعنى الدقيق للكلمة الدالة (تقوم بإرجاع قيمة) والإجراءات (لا تقوم بإرجاع قيمة) .

تركيب الجملة في بيثون لتعريف الدالة هو :

```
def nomDeLaFonction(liste de paramètres):
    ...
    bloc d'instructions
    ...
```

- يمكنك اختيار أي اسم للدالة التي تقوم بإنشائها، مع استثناء الكلمات المحجوزة للغة[22]، يجب عليك أن لا تستعمل أي رموز (باستثناء هذه _) كما هو الحال لأسماء المتغيرات، وأنصح باستخدام الأحرف الصغيرة في معظم الأوقات، وفي بداية اسم المتغير (الأسماء التي تبدأ بحرف كبير محجوز للصفوف والتي سوف نتحدث عنها لاحقا) .
- مثل العبارات **if** و **while** لتي قد عرفتها سابقا، والعبارة **def** هي عبارة مجمع. السطر الذي يحتوي على هذه العبارة ينتهي بالضرورة بنقطتين، والذي يحتوي على مجموعة من التعليمات التي يجب أن لا تنسهى فيها مسافة البادئة .
- قائمة البرامترات هي المعلومات التي تريد إعطاءها للدالة لاستعمالها (القوسين قد يكونا فارغين إذا كانت الدالة لا تحتاج إلى بارمترات)
- يتم استخدام الدالة مثل أي تعليمة تقريبا. في قلب البرنامج. يتم استدعاء الدالة عن طريق اسمها يليها قوسان. وإذا كان ضروريا، يتم وضع البارمترات التي تريد أن تحيلها إلى الدالة. ويجب عادة إدخال بارامتر واحد في تعريف الدالة، على الرغم من أنه يمكنك أن تحدد القيم الافتراضية لهذه البرامترات(سنتعرف إلى هذا لاحقا)

دالة بسيطة بدون برامترات

من أجل عمل برنامجنا الأول بالتطبيق العلمي للدوال ، سوف نستعمل مرة أخرى بيثون في الوضع التبادلي. الوضع التبادلي لبيثون هو في الواقع المثالي إجراء اختبارات صغيرة مثل التالية. هذه الميزة لا تجدها في جميع لغات البرمجة !

```
>>> def table7():
...     n = 1
...     while n <11 :
...         print(n * 7, end =' ')
...         n = n +1
...
```

عن طريق إدخال هذه الأسطر القليلة، عرفنا وظيفة بسيطة جدا وهي تحسب وتظهر أول عشرة نتائج لجدول الضرب على سبعة. لاحظ جيدا الأقواس[23]، النقطتين، والعبارة ومسافة البادئة (هي الكتلة التي تشكل جسم الدوال نفسه) .

لاستخدام دالة محددة أنشأناها، يمكننا استدعاءها بواسطة اسمها :

22تجد قائمة الكلمات المحجوزة في صفحة 13.

23اسم الدالة يجب أن يكون دائما مصحوبا بقوسين. حتى لو كانت الدالة لا تأخذ أي برامترات. وهذه هي نتيجة اتفاقية كتابة تنص على أن في أي نص يتعامل مع برمجة الحاسوب. يجب أن يصاحب اسم الدالة قوسين فارغين. ونحن نلتزم بهذه الاتفاقية في النص التالي .

```
>>> table7()
```

مما يؤدي إلى عرض :

```
7 14 21 28 35 42 49 56 63 70
```

يمكننا الآن إعادة استخدام هذه الدالة مرارا وتكرارا، مرات عديدة كما نرغب. يمكننا أيضا دمجها في تعرف دالة أخرى ، كما في المثال التالي :

```
>>> def table7triple():
...     print('La table par 7 en triple exemplaire :')
...     table7()
...     table7()
...     table7()
...
```

يمكننا استخدام هذه الميزة الجديدة عن طريق إدخال الأمر :

```
>>> table7triple()
```

عرض الناتج سيكون كالتالي :

```
La table par 7 en triple exemplaire :
7 14 21 28 35 42 49 56 63 70
7 14 21 28 35 42 49 56 63 70
7 14 21 28 35 42 49 56 63 70
```

و يمكن للدالة الثانية استدعاء الأولى ونفس الشيء مع الدالة الثالثة وإلخ ... في هذه المرحلة التي وصلنا إليها، قد لا تعرف بعد ما فائدة هذه ولكن يمكن ملاحظة فائدتين:

- إنشاء دالة جديدة تسمح لنا بإعطاء اسم لمجموعة من التعليمات. وبهذه الطريقة، يمكنك تبسيط الجسم الرئيسي للبرنامج، عن طريق إخفاء خوارزمية ثانوية معقدة تحت أمر واحد، والتي يمكن إعطاؤها اسما واضحا جدا، بالفرنسية إذا أردت .
- صنع دالة جديد يمكننا من إنشاء تصغير حجم البرنامج، من خلال إزالة الأجزاء المتكررة. على سبيل المثال، إذا أردت إظهار جدول سبعة مرات في نفس البرنامج، يمكنك أن لا تكتب كل مرة الخوارزمية التي تقوم بهذا العمل .

الدالة هي تعليمات جديدة خاصة بك، تستطيع إضافتها بحرية إلى لغة البرمجة الخاصة بك .

دالة مع بارمترات

في الأمثلة السابقة، لقد عرفنا واستعملنا دالة تظهر جدول الضرب 7. الآن نفترض أنه يجب علينا أن نفعل نفس الشيء مع جدول 9. هل يجب علينا أن نصنع دوال جديد لهذا ؟ ولنفترض أيضا أننا أردنا بعد ذلك أن نفعل نفس الشيء مع جدول 13 وهل يجب علينا البدء من جديد. أليس من الأفضل أن نعرف دالة قادرة على عرض أي جدول، على الطلب ؟

عندما نسمي هذه الدالة، يجب أن نكون طبعا قادرين على الإشارة إلى أي جدول نريد عرضه. هذه المعلومات يجب تمريرها للدالة والتي تسمى بارمتر. لقد التقينا عدة مرات مع دالات متكاملة تأخذ بارمتر. الدالة **sin(a)** - على سبيل المثال ، يحسب جوف الزاوية **a**. الدالة **sin()** استعملت إذا قيمة عددية كبرامتر للقيام بالعمل .

في تعريف الدالة، يجب أن يكون هنالك متغير خاص لتلقي البارامتر. هذا المتغير يسمى بارمتر. نختار له اسما لبناء قواعد التعليمة كالعادة، ونضع الاسم في ما بين قوسين المصاحبة لتعريف الدالة .

هذا الذي يجب كتابته في حالتنا :

```
>>> def table(base):
...     n = 1
...     while n <11 :
...        print(n * base, end =' ')
...        n = n +1
```

الدالة **table()** كما هو معرف أعلاه تستعمل البرامتر **base** لحساب أول عشرة نتائج لجدول الضرب الموافق .

اختبار هذه الميزة الجديدة، يجب علينا استدعاؤها مع البارامتر. مثال على ذلك :

```
>>> table(13)
13 26 39 52 65 78 91 104 117 130

>>> table(9)
9 18 27 36 45 54 63 72 81 90
```

في هذه الأمثلة، يتم تلقائيا تعيين قيمة بين قوسين عندما نستدعي الدالة (و بالتالي فهو بارامتر) إلى البرامتر القاعدة.

في جسم الدالة، **base**. تلعب دورا ليس كأي متغير آخر. عندما ندخل الأمر `table(9)`، نحن هنا نقصد الألة التي تنفذ الدالة **table()** عن طريق إعداد القاعدة لمتغير 9 للمتغير **base**.

استعمال المتغير كبرامتر

في المثالين أعلاه، البرامتر الذي استخدمناه للدالة **table()** وفي كل مرة عدد ثابت (المتغير 13 ثم المتغير 9). هذا ليس إلزاميا. لكن البرامتر الذي استخدمناه عند استدعاء الدالة يمكن أن يكون متغيرًا أيضا، مثل المثال أدناه. حلل المثال جيدا، حاول تشغيله، وصف في كراس التمارين الخاص بك ما يحدث. موضحا بشكل جيد. هذا المثال يعطيك دراسة أولية لاستعمال الدالات للقيام ببساطة مهام معقدة :

```
>>> a = 1
>>> while a <20:
...     table(a)
...     a = a +1
...
```

ملاحظة مهمة

في المثال أعلاه، البرامتر الذي مررناه لـ **table()** هو القيمة التي يحتويها المتغير **a**. داخل الدالة، هذا البرامتر يتم تعيينه لبرماتر **base**، الذي هو متغير آخر. إذا لاحظ الآن أن :

> اسم المتغير الذي تم تمريره كبرامتر ليس له أي علاقة باسم البرامتر المقابل في الدالة .

قد تكون هذه الأسماء هي نفسها إذا أردت، لكن يجب أن تعرف أنها لا تعبر عن نفس الشيء (على الرغم من أنها قد تحتوي على نفس القيمة الاختيارية) .

تمرين

1.7 قم باستدعاء وحدة turtle لأداء رسوم بسيطة. يمكنك رسم مجموعة من المثلثات متساوية الأضلاع مختلفة الألوان. وللقيام بذلك، يجب عليك تعريف متغير **triangle()** الذي يقوم برسم مثلث بلون محدد (و هذا يعني أن تعريف الدالة يجب أن يحتوي على برامتر لأخذ اللون). ثم استعمل هذه الدالة لرسم المثلث نفسه في أماكن مختلفة، مع تغيير اللون كل مرة .

دالة مع عدة برامترات

الدالة **table()** ثيرة للاهتمام بالتأكيد، لكنها لا تظهر سوى أول عشرة نتائج لجدول الضرب، ربما نرغب أن نظهر أكثر من ذلك. الآن، سوف نحسن الكود بإضافة برامترات أخرى في النسخة الجديدة التي سنطلق عليها هذه المرة **tableMulti()** :

```
>>> def tableMulti(base, debut, fin):
...     print('Fragment de la table de multiplication par', base, ':')
...     n = debut
...     while n <= fin :
...         print(n, 'x', base, '=', n * base)
...         n = n +1
```

هذه الميزة الجديدة سوف تستخدم ثلاثة برامترات : الأولى هي قاعدة الجدول مثل المثال السابق، الثانية هي أول عدد ضرب يبدأ به ، والثالث هي آخر عدد ضرب ينتهي به .

جرب هذه الدالة عن طريق إدخال هذا على سبيل المثال :

```
>>> tableMulti(8, 13, 17)
```

سوف يظهر :

```
Fragment de la table de multiplication par 8 :
13 x 8 = 104
14 x 8 = 112
15 x 8 = 120
```

```
16 x 8 = 128
17 x 8 = 136
```

ملاحظات

- لتحديد دالة مع عدة برامترات ، يجب عليك كتابتها في ما بين القوسين التي تلي اسم الدالة، مفصولة بفواصل .
- عند استدعاء الدالة، يجب أن يكون البرامترات في نفس أماكنها الصحيحة (و يتم فصلها أيضا بفاصلة). سيتم تعيين البرامتر الأول للبرامتر الاول للدالة ، والبرامتر الثاني للبرامتر الثاني في الدالة وإلخ ...
- كتمرين، حاول تسلسل التعليمات التالية وصف في كراس التمارين النتيجة :

```
>>> t, d, f = 11, 5, 10
>>> while t<21:
...     tableMulti(t,d,f)
...     t, d, f = t +1, d +3, f +5
...
```

متغير محلي، متغير عام

عندما نحدد متغيرات داخل جسم الدالة، هذه المتغيرات هي فقط للوصول إلى الدالة نفسها. نقول أن هذه المتغيرات محلية للدالة. هذا على سبيل المثال حالة من المتغيرات **fin**، **debut**، **base** و **n** للتمرين السابق.

في كل مرة يتم فيها استدعاء **tableMulti()** بيثون تحجز لها (في ذاكرة الحاسوب) مساحة جديدة للاسم[24]. محتويات المتغيرات **fin**، **debut**، **base** و **n** يتم تخزينها في مساحة الاسم التي لا يمكن الوصول إليها من خارج الدالة. على سبيل المثال، إذا أردت إظهار محتوى الدالة **base** بعد القيام بالتمرين السابق، سوف تحصل على رسالة الخطأ التالية :

```
>>> print(base)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'base' is not defined
```

الآلة تقول لنا بوضوح أن الرمز **base** غير معروف، في حين أنه تم طباعته من قبل الدالة **tableMulti()** نفسها. مساحة الأسماء التي يحتويها الرمز **base** مقتصرة فقط للدالة **tableMulti()**، ويتم حذفها تلقائيا عندما تنتهي الدالة من عملها .

المتغيرات المعرفة خارج الدالة هي متغيرات عامة. محتواها " مقروء " للدالة، لكن الدالة لا تستطيع تغيير محتواها. على سبيل المثال :

---

24 هذا المفهوم لمساحة الأسماء سوف نتعمق به تدريجيا. وسوف نتعلم أيضا في وقت لاحق أن الدالات هي في الواقع كائنات يتم إنشاء نسخة جديدة منها كلما تم استدعاؤها .

```
>>> def mask():
...     p = 20
...     print(p, q)
...
>>> p, q = 15, 38
>>> mask()
20 38
>>> print(p, q)
15 38
```

حلل بعناية هذا المثال :

سوف نبدأ من خلال تعريف دالة بسيطة جدا (و التي لا تستعمل أي برامترات). في هذه الدالة، يتم تعريف الدالة **p** مع القيمة الأولية لهذا المتغير هي **20** هذا المتغير سوف يكون داخل الدالة أي محلي .

بمجرد الانتهاء من تعريف الدالة، نعود للمستوى الأساسي ونعرف متغيرين **p** و **q** الذان يحتويان على القيمتين **15** و **38**. هذان المتغيران يتم تعريفهما في المستوى الأساسي سيكونان إذا متغيرات عامة .

و هكذا تم استخدام نفس المتغير **p** مرتين، لتحديد اثنين من المتغيرات المختلفة : واحد عام والثاني محلي. يمكن النظر إلى هذين المتغيرين على أنهما متغيران مستقلان ومنفصلان عن بعضهما البعض، حسب القاعدة فإن في الدالة (حيث يكون التنافس)، يكون للمتغيرات المحلية أولوية .

في الواقع يبدو أنه عندما يتم تشغيل دالة **mask()** المتغيران العامان **q** و لا يبدو أنه يمكن الوصول إليهما، منذ يتم طباعتهم بشكل جيد، لـ **p**، على العكس، القيمة المحلية هي التي يتم إظهارها ..

قد يعتقد المرء في البداية أن دالة **mask()** ببساطة تغير محتوى المتغير العام **p** (بما أنه يمكن الوصول إليه). الأسطر التالية تظهر أنه ليس كذلك : عند الخروج من الدالة **mask()**، يعود المتغير العام **p** إلى قيمته الأولية .

يبدو هذا كله معقدا في البداية، لكن سرعان ما سنعرف كم هو مفيد تعريف المتغيرات كمحلية، وهذا معناه بطريقة أخرى أنها تقتصر فقط على الدالة. هذا يعني أنك تستطيع دائما استخدام عدد لا نهائي من الدالات دون الحاجة إلى القلق من أسمائها سواء أكانت مستخدمة سابقا أو لا : هذه المتغيرات لا تستطيع أبدا أن تتداخل مع تلك التي عرفتها بنفسك في مكان آخر .

تستطيع تغيير هذا إذا أردت. لعلك على سبيل المثال قد عرفت الدالة التي يجب عليها تغيير محتوى متغير عام. لفعل هذا، ببساطة استعمل التعليمة **global**. هذه التعليمة تمكنك من الإشارة داخل تعريف الدالة - تمكنك من التعامل مع المتغيرات بشكل شامل .

في المثال أدناه، يستخدم المتغير **a** داخل الدالة **monter()** ليس فقط للوصول، بل حتى تغيير محتواها، لأنه تم تعريفه على أنه متغير باستعمال شامل. وعلى سبيل المقارنة، قم بعمل نفس التمرين لكن هذه المرة احذف التعليمة **global** : المتغير لا يزداد مع كل استدعاء للدالة .

```
>>> def monter():
...     global a
...     a = a+1
...     print(a)
...
>>> a = 15
>>> monter()
16
>>> monter()
17
>>>
```

الدالات الحقيقية والإجراءات

للمتمرسين، الدالات التي وصفناها هنا هي في المعنى الدقيق للكلمة ليست كذلك، ولكن بشكل أكثر دقة الإجراءات[25]. الدالة "الحقيقية" (بالمعنى الدقيق) يجب عليها أن تعود قيمة واحدة عندما تنتهي. الدالة "الحقيقية" تستطيع استعمال علامة المساواة في التعبيرات مثل `y = sin(a)`. نفهم من هذا أن هذه العبارة، الدالة **sin()** تعيد قيمة (في داخل البرامتر **a**) الذي تم تعيينه مباشرة إلى المتغير **y**.

دعونا نبدأ مع مثال بسيط للغاية :

```
>>> def cube(w):
...     return w*w*w
...
```

العبارة **return** تحدد القيمة التي يجب إرجاعها من الدالة. في هذه الحالة، هذا هو مكعب البرامتر الذي تم تمريره عند استدعاء الوظيفة. على سبيل المثال :

```
>>> b = cube(9)
>>> print(b)
729
```

على سبيل المثال أكثر قليلا تعقيدا، سوف نقوم الآن بتغيير صغير على دالة **table()** الذي عملنا به عمل لا بأس به، سوف نجعله يعود بقيمة. هذه القيمة في هذه الحالة هي قائمة (قائمة متكونة من أول عشرة نتائج لجدول الضرب المختار. هذه فرصة جيدة لإعادة الحديث عن القوائم. في هذه العملية، يجب علينا أن ننتهز هذه الفرصة لتعلم شيئا جديدا .

---

25 في بعض لغات البرمجة، يتم تعريف المهام والإجراءات باستخدام تعليمات مختلفة، بيثون يستخدم التعليمة `def` pour définir les unes et les autres.

```
>>> def table(base):
...     resultat = []           # النتيجة الأولى هي قائمة فارغة
...     n = 1
...     while n < 11:
...        b = n * base
...        resultat.append(b)   # إضافة قيمة إلى القائمة
...        n = n +1             # (انظر إلى الشرح أدناه)
...     return resultat
...
```

لتجربة هذه الدالة، نستطيع أن ندخل على سبيل المثال :

```
>>> ta9 = table(9)
```

و بالتالي نحن خصصنا للمتغير **ta9** أول عشرة نتائج لجدول الضرب على 9، في شكل لائحة :

```
>>> print(ta9)
[9, 18, 27, 36, 45, 54, 63, 72, 81, 90]
>>> print(ta9[0])
9
>>> print(ta9[3])
36
>>> print(ta9[2:5])
[27, 36, 45]
>>>
```

(تذكر: العنصر الأول في الدالة هو المؤشر 0)

ملاحظات

- كما رأينا في المقال السابق، العبارة **return** تحدد ما هي القيمة التي يجب أن تعود من الدالة. في هذه الحالة، هذا هنا هو محتوى المتغير **resultat**، هذا معناه قائمة الأرقام التي تم صناعتها من قبل الدالة[26].
- العبارة **resultat.append(b)** هي المثال الثاني لنا لاستخدام مفهوم الاستدعاء الذي لا يزال الكثير من الأشياء منه لم نضعها : هذه العبارة، نحن نطبق طريقة **append()** للكائن **resultat**.

سوف نشرح الآن خطوة بخطوة ما المقصود ببرمجة الكائن. الآن، نعترف ببساطة بأن هذا المصطلح عام جدا ينطبق بشكل خاص على قوائم بيثون. الأسلوب ليس أكثر من مجرد دالة (و يمكنكم أيضا معرفتها بوجود الأقواس)، لكن هنا الدالة مرتبطة بكائن. وهي جزء من تعريف هذا الكائن، أو بشكل أكثر دقة فئة معينة تنتمي لهذا الكائن (سوف ندرس مفهوم الطبقة في وقت لاحق).

[26] يمكن استخدام `return` بدون أي برامتر داخل الدالة، مما يتسبب في إغلاق البرنامج مباشرة. والقيم التي يتم إرجاعها في هذه الحالة كائن `None` (كائن خاص، "لا شيء") .

يتم تنفيذ الأسلوب المرتبط بالكائن بطريقة أو بأخرى "بتشغيل دالة" هذا الكائن بطريقة معينة. على سبيل المثال يتم تطبيق الأسلوب **methode4()** لكائن **objet3** بمساعدة تعليمة النوع : **objet3.methode4()** ، وهذا معناه اسم الكائن، ثم اسم الأسلوب، متصلة ببعضها البعض بواسطة نقطة. هذه النقطة لها دور أساسي : يمكن اعتبارها معاملا حقيقيا.

في مثالنا، نحن نطبق الأسلوب **append()** إلى كائن **resultat** الذي هو قائمة. في بيثون، القوائم هي فئة معينة من الكائنات، يمكن أن تطبق بشكل فعال على مجموعة متنوعة من الأساليب. في هذه الحالة، الأسلوب **append()** مجموعة كائنات "قوائم" يستخدم لإضافة عنصر إلى النهاية. الكائن الذي سيتم إضافته سيكون داخل أقواس، مثل جميع البرامترات .

- كنا قد حصلنا على نتيجة مماثلة إذا استخدمنا بدلا من هذه العبارة التعليمة " `resultat = resultat + [b]` " (معامل السلسلة يعمل في الواقع أيضا مع القوائم)، هذه الطريقة هي أقل كفاءة وفعالية، لأنها تقوم بإعادة تعريف قائمة جديدة عند كل تكرار جديد للحلقة. حيث القائمة الكاملة السابقة تقوم بكل مرة بإعادة نسخها مع إضافة عنصر إضافي. من سلبيات استخدام أسلوب **append()**، أن الحاسوب يقوم في الواقع بتعديل قائمة موجودة بالفعل (دون نسخه في متغير جديد). إذا هذا الأسلوب هو الأفضل، لأنه يستخدم موارد أقل، بالإضافة إلى أنه أسرع (وخاصة عند التعامل مع القوائم الكبيرة) .
- ليس لزامًا علينا أن كل قيمة يتم إرجاعها من دالة يجب أن تكون بواسطة المتغير (كما فعلنا حتى الآن في هذه الأمثلة). وبالتالي، يمكننا اختبار الدالتين **cube()** و **table()** عن طريق إدحال هذين الأمرين :

```
>>> print(cube(9))
>>> print(table(9))
>>> print(table(9)[3])
```

أو بشكل أكثر بساطة :

```
>>> cube(9)...
```

استعمال الدالات داخل سكريبت

ومن أجل هذا الأسلوب الأول من الدالات، قمنا فيما سبق باستخدام الوضع التفاعلي لبيثون.

فمن الواضح أنه يمكننا استخدام الدالات في البرامج النصية (سكريبت) كذلك. الرجاء حاول فعل ذلك بنفسك مع البرنامج الصغير في الأسفل، والتي تحسب حجم الكرة باستخدام الصيغة التي ربما قد تعرفها : $V=\frac{4}{3}\pi R^3$

```
def cube(n):
   return n**3

def volumeSphere(r):
   return 4 * 3.1416 * cube(r) / 3

r = input('Entrez la valeur du rayon : ')
print('Le volume de cette sphère vaut', volumeSphere(float(r)))
```

ملاحظات

بنظرة أقرب، يتكون هذا البرنامج من ثلاثة أجزاء رئيسية : الدالتان **cube()** و **volumeSphere()**، والجسم الأساسي للبرنامج.

في الجسم الأساسي في البرنامج، استدعينا الدالة **volumeSphere()**، وسوف نمرر لها القيمة المدخلة من قبل المستخدم لقطر نصف الدائرة، ويتم تحويلها إلى عدد حقيقي بمساعدة الدالة المدمجة **float()**.

داخل الدالة **volumeSphere()**، هنالك استدعاء للدالة **cube()**.

لاحظ أن الأجزاء الثلاثة في البرنامج مرتبة بترتيب معين : أولا نبدأ بتعريف الدالات، ثم نقوم بكتابة الجزء الأساسي للبرنامج. هذا الترتيب ضروري، لأن المفسر يقوم بتنفيذ أسطر التعليمات واحدة تلو الأخرى، وفقا لترتيب ظهورها في السورس كود (شفرة البرنامج) .

> في السكريبت، يجب أن يكون تعريف الدالات سابقا لاستخدامها (يجب عليك تعريفها في البداية).

لتقتنعوا، اعكس هذا النظام (على سبيل المثال، ضع جسم البرنامج في البداية)، ثم لاحظ ظهور رسالة خطأ عند تشغيل السكريبت المعدل.

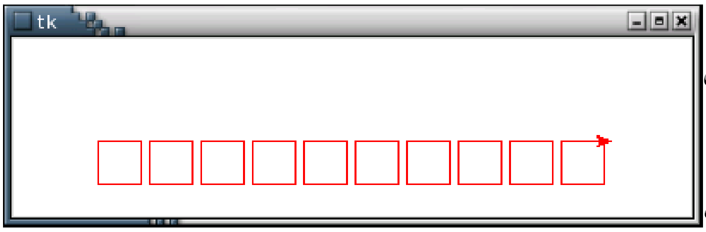
في الواقع، الجسم الأساسي للبرنامج المكتوب ببيثون هو جزء خاص إلى حد ما، الذي يعرف دائما عند الأعمال الداخلية للمفسر تحت الاسم المحجوز **__main__** (الكلمة "main" تعني أساسي باللغة الأنكليزية. يكون الاسم محدد بعلامتي خط على جانبي الاسم، لتجنب الخلط بينه وبين غيره من الرموز). وعند تشغيل السكريبت، يبدأ دائما بعبارة هذا الجزء **__main__**، وقد يكون هذا موجودا في القائمة. ويتم تنفيذ هذه التعليمات واحدة تلو الأخرى، وذلك حتى يتم استدعاء الدالة الأولى. عند استدعاء دالة يكون مثل التفاف في تدفق تنفيذ البرنامج : بدلا من الانتقال إلى التعليمة التالية، يقوم المفسر بتنفيذ الدالة التي تم استدعاؤها، ثم يعود البرنامج إلى السطر الذي كان فيه ليكمل العمل الذي انقطع عنه. لهذه الآلية في العمل، يجب أن يكون المفسر قادرا على قراءة وتعرف الوظيفة قبل **__main__**، وهذا الأخير يجب وضعه في نهاية سكريبت البرنامج.

في مثالنا، جزء __a __mainيستدعي الدالة الأولى وهي تستدعي الدالة الثانية. هذه الطريقة شائعة جدا في البرمجة. إذا أردت أن تفهم بشكل صحيح ماذا يحدث في أحد البرامج، يجب عليك إذا تعلم قراءة السكريبت، ليس من السطر الأول إلى السطر الأخير، لكن باتباع مسار مشابه لما يحدث عند تشغيل البرنامج. هذا يعني بالضبط أنه يجب عليك تحليل السكريبت بدأ من الأسطر الأخيرة !

وحدات الدالات

وحتى تتمكن جيدا من التمييز بين تعريف دالة واستخدامها في البرنامج، نقترح عليك في الكثير من الأحيان وضع تعريفات الدالات في وحدة بيثون، والبرنامج الذي يستخدمها في مكان آخر .

مثال :



مطلوب منك إنتاج سلسلة من الرسوم على الجانب، وذلك بمساعدة وحدة turtle :

اكتب الأسطر البرمجية التالية، وقم بحفظها في ملف باسم ***dessins_tortue.py*** :

```
from turtle import *

def carre(taille, couleur):
    "fonction qui dessine un carré de taille et de couleur déterminées"
    color(couleur)
    c =0
    while c <4:
        forward(taille)
        right(90)
        c = c +1
```

قد تلاحظ أن تعريف الدالة **carre()** يبدأ بسلسلة نصية. هذه السلسلة لا تلعب أي دور وظيفي في السكريبت : تتم معالجة هذه السلسلة كتعليق بسيط من قبل بيثون، لكن يتم تخزينها داخل جزء نظام الوثائق الداخلية التلقائي ، ثم يمكن استغلالها من قبل المستخدمين والناشرين (ذكي).

إذا كنت في بيئة IDLE، على سبيل المثال، تستطيع هذه السلسلة النصية التوثيق "تلميح"، تستطيع في كل مرة يتم استدعاء دالات موثقة جيدا.

في الواقع، بيثون يضع هذه السلسلة النصية داخل متغير خاص تحت اسم **__doc__** (الكلمة "doc" بجانبها خطان من كل جهة)، ويرتبط بكائن دالة مثل خصائصه (سوف تتعلم عن هذه الصفات عندما نناقش طبقات الكائنات صفحة 178). ولتتمكن من العثور على سلسلة التوثيق لدالة معينة أعرض محتوى هذ المتغير. مثال :

```
>>> def essai():
...     "Cette fonction est bien documentée mais ne fait presque rien."
...     print("rien à signaler")
...
>>> essai()
rien à signaler
>>> print(essai.__doc__)
Cette fonction est bien documentée mais ne fait presque rien.
```

خذ إذا عناء كتابة السلاسل وابذل كل جهدك لتعرف الدالات في المستقبل : هذه الممارسة موصى بها كثيرا. الملف الذي صنعته الآن هو وحدة بيثون صحيحة، تماما مثل الوحدات turtle أو math التي قد عرفتها في وقت سابق. تستطيع الآن استعمالها في أي سكريبت آخر، مثل هذا على سبيل المثال، يؤدي العمل المطلوب :

```
from dessins_tortue import *

up()                    # إزالة القلم
goto(-150, 50)          # العودة إلى أعلى اليسار

# رسم عشرة مربعات حمراء بمحاذات بعضها البعض :
i = 0
while i < 10:
    down()              # إنزال القلم
    carre(25, 'red')    # رسم المربع
    up()                # إزالة القلم
    forward(30)         # الابتعاد
    i = i +1
a = input()             # الانتظار
```

> انتبه
>
> يمكنك تسمية وحدات دالاتك على النحو الذي تراه مناسبا. ولكن يجب أن تدرك أنه لا يمكن استدعاء وحدة إذا كان اسمها محجوزا لبيثون (الموجودة في الصفحة 13، لأن اسم الوحدة المستدعاة ستصبح متغيرًا في سكريبتك. والكلمات المحجوزة لا يمكن أن تستخدم كأسماء متغيرات. تذكر أيضا أنه لا يمكنك إعطاء اسم لوحداتك (و لكل سكريبتاتك بصفة عامة) بنفس اسم لوحدة موجودة في بيثون . وإلا سوف تحدث مشاكل. على سبيل المثال، إذا أعطيت اسم **turtle.py** لتمرين وضعت فيه تعليمة لاستدعاء وحدة **turtle**. فسيتم استدعاء هذا التمرين نفسه !

Résumé : structure d'un programme Python type

```
# -*- coding:Utf8 -*-

####################################
# Programme Python type            #
# auteur : G.Swinnen, Liège, 2009  #
# licence : GPL                    #
####################################

#####################################
# Importation de fonctions externes :

from math import sqrt

##################################
# Définition locale de fonctions :

def occurrences(car, ch):
    "Cette fonction renvoie le \
     nombre de caractères <car> \
     contenus dans la chaîne <ch>"

    nc = 0

    i = 0

    while i < len(ch):

        if ch[i] == car:
            nc = nc + 1

        i = i + 1

    return nc

################################
# Corps principal du programme :

print("Veuillez entrer un nombre :")
nbr = eval(input())

print("Veuillez entrer une phrase :")
phr = input()
print("Entrez le caractère à compter :")
cch = input()

no = occurrences(cch, phr)
rc = sqrt(nbr**3)

print("La racine carrée du cube", end=' ')
print("du nombre fourni vaut", end=' ')
print(rc)

print("La phrase contient", end=' ')
print(no, "caractères", cch)
```

Un programme Python contient en général les blocs suivants, dans l'ordre :
- Quelques instructions d'initialisation (importation de fonctions et/ou de classes, définition éventuelle de variables globales).
- Les définitions locales de fonctions et/ou de classes.
- Le corps principal du programme.

Le programme peut utiliser un nombre quelconque de fonctions, lesquelles sont définies localement ou importées depuis des modules externes.
Vous pouvez vous-même définir de tels modules.

La définition d'une fonction comporte souvent une liste de PARAMÈTRES.
Ce sont toujours des VARIABLES, qui recevront leur valeur lorsque la fonction sera appelée.

Une boucle de répétition de type 'while' doit toujours inclure au moins quatre éléments :
- l'initialisation d'une variable 'compteur' ;
- l'instruction while proprement dite, dans laquelle on exprime la condition de répétition des instructions qui suivent ;
- le bloc d'instructions à répéter ;
- une instruction d'incrémentation du compteur.

La fonction "renvoie" toujours une valeur bien déterminée au programme appelant.
Si l'instruction 'return' n'est pas utilisée, ou si elle est utilisée sans argument, la fonction renvoie un objet vide : 'None'.

Le programme qui fait appel à une fonction lui transmet d'habitude une série d'ARGUMENTS, lesquels peuvent être des valeurs, des variables, ou même des expressions.

تمارين

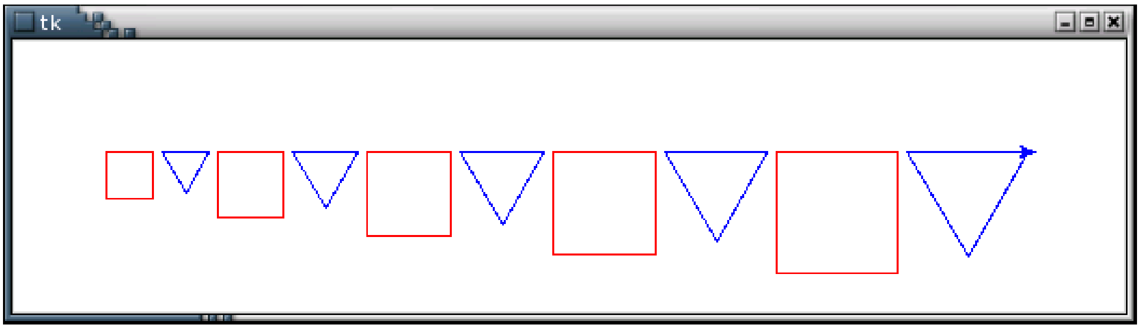
2.7 عرف الدالة **ligneCar(n, ca)** التي تقوم بإرجاع سلسلة نصية n لـ **ca**.

3.7 عرف الدالة **surfCercle(R)**. هذه الدالة تقوم بإرجاع السطح (المنطقة) لدائرة قدمنا لك قطرها **R** في برامتر.
على سبيل المثال، عند تنفيذ التعليمة :
`print(surfCercle(2.5))` يجب أن يكون الناتج : 19.63495...

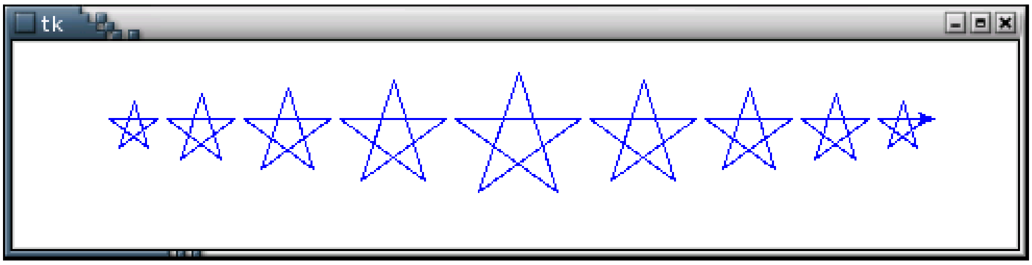
4.7 عرف الدالة **volBoite(x1,x2,x3)** التي تقوم بإرجاع حجم علبة متوازية تم وضع أبعادها الثلاثة **x1, x2, x3** كبرامتر .
على سبيل المثال، عند تنفيذ التعليمة :
`print(volBoite(5.2, 7.7, 3.3))` يجب أن يكون الناتج : 132.132.

5.7 عرف الدالة **maximum(n1,n2,n3)** التي تقوم بإرجاع أكبر عدد بين 3 أعداد **n1, n2, n3** التي هي في البرامتر. على سبيل المثال عند تنفيذ التعليمة التالية
`print(maximum(2,5,4))` يجب أن يكون الناتج : 5.

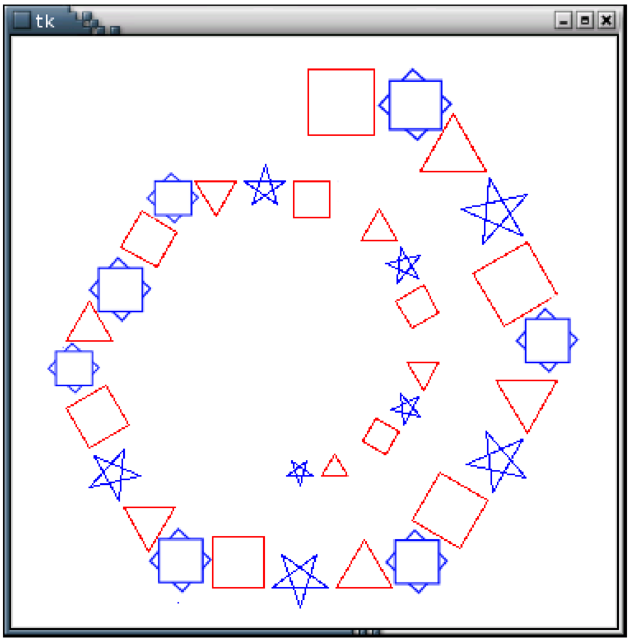
6.7 أكمل وحدة الدالات الرسومية **dessins_tortue.py** والتي تم وصفها في الصفحة 74.
ابدأ بإضافة البرامتر **angle** إلى دالة **carre()**، بحيث يمكن وضع المربعات في اتجاهات مختلفة .
ثم حدد الوظيفة **triangle(taille, couleur, angle)** القادرة على رسم مثلث متوازي الأضلاع بلون واتجاه موضوع كبرامتر. اختبر الوحدة الخاصة بك بمساعدة برنامج يقوم باستدعاء هذه الدالات عدة مرات، مع مجموعة من البرامترات المتنوعة التي تقوم برسم مجموعة مربعات ومثلثات :



7.7 أضف إلى وحدة التمرين السابق الدالة **etoile5()** المتخصصة برسم نجمة بخمسة أفرع. داخل البرنامج الأساسي، أضف حلقة ترسم مجموعة من 9 نجمات صغيرة بأحجام مختلفة :



8.7 أضف إلى وحدة التمرين السابق دالة **etoile8()** تقوم برسم نجمة بـ 8 أفرع، وهي تتكون من مربعين متداخلين في بعضهما البعض. هذه الدالة الجديدة تقوم باستدعاء الدالة **carre()** التي تم تعريفها سابقا. وبرنامجك يجب أن يقوم برسم سلسلة من هذه النجوم :



9.7 عرف الدالة **compteCar(ca,ch)** التي تقوم بإرجاع عدد مرات التي يتكرر فيها الحرف **ca** داخل السلسلة النصية **ch**. على سبيل المثال، عند تنفيذ التعليمة :

`print(compteCar('e', 'Cette phrase est un exemple'))` يعطينا الناتج : 7

10.7 عرف الدالة **indexMax(liste)** التي تقوم بإرجاع مؤشر العنصر ذي القيمة الأعلى داخل السلسلة على شكل برامتر. مثال للتشغيل :

```
 serie = [5, 8, 2, 1, 9, 3, 6, 7]
print(indexMax(serie))
4
```

11.7 عرف الدالة **nomMois(n)** التي تقوم بإرجاع اسم شهر للسنة على سبيل المثال، عند تنفيذ التعليمة : `print(nomMois(4))` تقوم بإعطاء الناتج : : `Avril`.

12.7 عرف الدالة **inverse(ch)** التي تقوم بعكس ترتيب حروف في أي سلسلة. السلسلة المعكوسة سيتم إرجاعها للبرنامج الذي استدعى الدالة .

13.7 عرف الدالة **compteMots(ph)** التي تقوم بإرجاع عدد الكلمات التي تحتويها الجملة **ph**. ونعتبر الكلمة هي مجموعة من الحروف ويكون بين الكلمات مسافات .

كتابة البرامترات

لقد تعلمت كتابة المتغيرات في بيثون بشكل فعال، وهذا معناه أنه يتم تعريف نوع المتغير في نفس الوقت الذي تقوم بوضع قيمته. هذه الألية تعمل أيضا لبرامترات الدالة. نوع البرامتر يصبح تلقائيا عندما يتم تمرير القيمة كبرامتر للدالة. على سبيل المثال :

```
>>> def afficher3fois(arg):
...     print(arg, arg, arg)
...

>>> afficher3fois(5)
5 5 5

>>> afficher3fois('zut')
zut zut zut

>>> afficher3fois([5, 7])
[5, 7] [5, 7] [5, 7]

>>> afficher3fois(6**2)
36 36 36
```

في هذا المثال، قد تجد أن الدالة **afficher3fois()** تقبل جميع أنواع البرامترات التي يتم تمريرها على مختلف أنواعها، وهي رقم، سلسلة نصية، قائمة أو حتى تعبير. في الحالة الأخيرة، بيثون يقوم بفحص التعبير، ويقوم بتمرير ناتج عملية التعبير كبرامتر للدالة .

القيم الافتراضية للبرامترات

في تعريف الدالة، من الممكن (و مرغوب في الكثير من الأحيان) تعريف قيمة برامتر افتراضية لكل برامتر. وهذا يعطي الدالة التي نستطيع تسكينها مع مجموعة فقط من البرامترات المنتظرة. على سبيل المثال :

```
>>> def politesse(nom, vedette ='Monsieur'):
...    print("Veuillez agréer ,", vedette, nom, ", mes salutations cordiales.")
...
```

```
>>> politesse('Dupont')
Veuillez agréer , Monsieur Dupont , mes salutations cordiales.

>>> politesse('Durand', 'Mademoiselle')
Veuillez agréer , Mademoiselle Durand , mes salutations cordiales.
```

عند استدعاء هذه الدالة، القيمة الأولى قد وضعناها أما القيمة الثانية ستأخذ القيمة الافتراضية. وإذا أدخلنا قيمتين، القيمة الافتراضية الثانية سوف تلغى.

يمكن تعيين قيمة افتراضية لكن البرامترات، أو جزء منها فقط. في هذه الحالة ، ومع ذلك ، البرامترات بدون قيم يجب أن تسبق بقية القيم. على سبيل المثال، المثال في الأسفل غير صحيح :

```
>>> def politesse(vedette ='Monsieur', nom):
```

مثال آخر :

```
>>> def question(annonce, essais =4, please ='Oui ou non, s.v.p.!'):
...     while essais >0:
...         reponse = input(annonce)
...         if reponse in ('o', 'oui','O','Oui','OUI'):
...             return 1
...         if reponse in ('n','non','N','Non','NON'):
...             return 0
...         print(please)
...         essais = essais-1
...
>>>
```

يمكن استدعاء هذه الدالة بطرق مختلفة، على سبيل المثال :

```
rep = question('Voulez-vous vraiment terminer ? ')
```

أو :

```
rep = question('Faut-il effacer ce fichier ? ', 3)
```

أو :

```
rep = question('Avez-vous compris ? ', 2, 'Répondez par oui ou par non !')
```

خذ وقتا في تشريح هذا المثال .

برامترات مع علامات

في معظم لغات البرمجة، البرامترات التي نضعها عند استدعاء الدالة تكون في نفس مكانها في تعريف الوظيفة.

بيثون تسمح بقدر كبير من المرونة. إذا حصلت البرامترات في تعريفها في الدالة على قيمة ، كما هو موضح أعلاه، يمكننا استدعاء الدالة عن طريق تقديم البرامترات على أي ترتيب، على شرط أن نكتب اسم البرامتر بشكل صحيح، على سبيل المثال :

```
>>> def oiseau(voltage=100, etat='allumé', action='danser la java'):
...     print('Ce perroquet ne pourra pas', action)
...     print('si vous le branchez sur', voltage, 'volts !')
...     print("L'auteur de ceci est complètement", etat)
...

>>> oiseau(etat='givré', voltage=250, action='vous approuver')
Ce perroquet ne pourra pas vous approuver
si vous le branchez sur 250 volts !
L'auteur de ceci est complètement givré

>>> oiseau()
Ce perroquet ne pourra pas danser la java
si vous le branchez sur 100 volts !
L'auteur de ceci est complètement allumé
```

تمارين

14.7 عدل الدالة **volBoite(x1,x2,x3)** التي تم تعريفها في التمرين السابق، بحيث يمكن استدعاؤها ببراماتر واحد أو اثنين أو ثلاثة برامترات، أو بدون برامترات. استخدم القيم الافتراضية للقيم هي 10، على سبيل المثال :

`print(volBoite())` نتيجته : 1000

`print(volBoite(5.2))` نتيجته : 520.0

`print(volBoite(5.2, 3))` نتيجته : 156.0

15.7 عدل الدالة **volBoite(x1,x2,x3)** التي في الأعلى بطريقة بحيث يمكننا استدعاؤها مع برامتر واحد أو اثنين أو ثلاثة برامترات. في حالة استخدام برامتر واحد، يكون الصندوق على شكل مكعب (البرامترات يجب أن تعبر عن الحافة). إذا تم استخدام برامترين، يبدو كأنه مربع منشور (في هذه الحالة البرامتر الأولى للجانب والثانية لارتفاع المنشور). وإذا كانت ثلاثة، تكون على شكل متوازي، على سبيل المثال :

`print(volBoite())` (نتيجته : 1- (يشير إلى خطأ)

`print(volBoite(5.2))` نتيجته : 140.608

`print(volBoite(5.2, 3))` نتيجته : 81.12

`print(volBoite(5.2, 3, 7.4))` نتيجته : 115.44

16.7 عرف دالة **changeCar(ch,ca1,ca2,debut,fin)** التي تبدل كل حروف **ca1** بحروف **ca2** في سلسة نصية **ch**، بداية من المؤشر **debut** وإلى المؤشر **fin**، هذان البرامتران الأخيران يمكننا تركهما (وفي هذه الحالة يتم التعامل مع سلسلة واحدة من البداية إلى النهاية)، أمثلة على الدالة المتوقعة :

```
>>> phrase = 'Ceci est une toute petite phrase.'
>>> print(changeCar(phrase, ' ', '*'))
Ceci*est*une*toute*petite*phrase.
>>> print(changeCar(phrase, ' ', '*', 8, 12))
Ceci est*une*toute petite phrase.
>>> print(changeCar(phrase, ' ', '*', 12))
Ceci est une*toute*petite*phrase.
>>> print(changeCar(phrase, ' ', '*', fin = 12))
Ceci*est*une*toute petite phrase.
```

17.7 عرف الدالة **eleMax(liste,debut,fin)** التي تقوم بإرجاع القيمة الأعلى في السلسلة التي تم تمريرها ، البرامتران **debut** و **fin** يشيران إلى المؤشرات التي ينبغي البحث عنها، ويمكن حذفها (كما في التمرين السابق).
أمثلة على الدالة المتوقعة :

```
>>> serie = [9, 3, 6, 1, 7, 5, 4, 8, 2]
>>> print(eleMax(serie))
9
>>> print(eleMax(serie, 2, 5))
7
>>> print(eleMax(serie, 2))
8
>>> print(eleMax(serie, fin =3, debut =1))
6
```

8

استخدام النوافذ والرسومات

حتى الآن ، استخدمنا بيثون فقط في "الوضع النصي" لأنه يجب علينا أن نتعلم أولا عددًا من المفاهيم الأساسية والبنية الأساسية للغة، قبل أن نبدأ تعلم أشياء أكثر صعوبة وتطورا (مثل النوافذ والصور والأصوات، إلخ ...)يمكننا الآن التوغل في بيثون والدخول إلى حقل واسع من الواجهات الرسومية ، لكن هذا لن يكون سوى البداية : على الرغم من أننا لم نتعلم الكثير ومازال أمامنا الكثير من الأساسيات يجب أن نتعلمها، وربما أصبح "الوضع النصي " محبوبا لدى الكثير منكم .

واجهات المستخدم الرسومية (GUI)

إن كنت تجهل هذا حتى الآن ، اعلم أن مجال الواجهات الرسومية في غاية التعقيد والصعوبة. لكل نظام تشغيل يتوفر عدة "مكتبات" لوظائف الرسم الأساسية ، التي تضاف (في كثير من الأحيان) إلى العديد من المكملات ، (أكثر أو أقل بحسب لغات البرمجة) وتعرض جميع هذه المكونات بشكل عام فئات للكائن (كلاس أوبجيكت) والتي سندرس سماتها وأساليبها.

مع بيثون ، المكتبة الرسومية الأكثر استخداما حتى الآن (هذا الكتاب قديم) مكتبة تكنتر الذي هو تكييف لمكتبة تاكا وضعت أصلا للغة برمجة Tcl و wxPython : وهناك أيضا عدة مكتبات رسومية للغة برمجة بيثون مثل PyQT و Pygtk ...إلخ

و هناك إمكانية لاستخدام مكتبات جافا ومكتبات ميكروسوفت أم أف سي لنظام ويندوز. إضافة إلى هذا نحن سنتعلم فقط البرمجة باستخدام تكنتر التي توجد لحسن الحظ نسخ لعدة أنظمة تشغيل (وبشكل مجاني) منها ويندوز وليكس وماك

الخطوات الأولى مع Tkinter

للمزيد من الإيضاح ، نحن نفترض بالطبع أن وحدة Tkinter[27] مثبتة مسبقا على نظامك. لتكون قادرا على استخدام مميزات تكنتر يجب عليك أن تستدعيه (بسطر واحد فقط) بإضافة هذا السطر إلى ملف البرنامج :

```
from tkinter import *
```

[27] في إصدارات بيثون السابقة (قبل الإصدار الثالث) تبدأ اسم الوحدة بحرف كبير



كالعادة ، ليس من الضروري على بيثون كتابة سكريبت بل تستطيع فعل هذا من خلال سطر الأوامر (بعد تشغيل بيثون) في مثالنا التالي سوف نقوم بإنشاء نافذة بسيطة ، ثم نضيف فيها أداتين[28]، أداة جزء من النص (عنوان) وزر.

```
>>> from tkinter import *
>>> fen1 = Tk()
>>> tex1 = Label(fen1, text='Bonjour tout le monde !', fg='red')
>>> tex1.pack()
>>> bou1 = Button(fen1, text='Quitter', command = fen1.destroy)
>>> bou1.pack()
>>> fen1.mainloop()
```

> اعتمادا على هذه النسخة من بيثون ، سوف نرى نافذة التطبيق تظهر مباشرة بعد إدخال الأمر الثاني في مثالنا هذا أو بعد السطر السابع فقط[29].

دعونا الآن نبحث عن المزيد في كل أسطر الأوامر المنفذة

1. كما سبق شرحه أعلاه، فإنه من السهل بناء وحدات بيثون المختلفة، والتي تحتوي على سكريبتات، تعريفات الدالات، أصناف الكائنات، إلخ ... يمكننا إذا استدعاء جزء أو كل من هذه الوحدات لأي برنامج، حتى لو كنا داخل مفسر يعمل بالوضع التفاعلي(هذا معناه مباشرة إلى سطر الأوامر). هذا ما فعلناه في السطر الأول لمثالنا :

   `from tkinter import *` معناه استدعاء جميع الأصناف في وحدة tkinter.

2. سيكون لدينا المزيد حول هذه الفئات. في البرمجة، تسمي مولدات الكائنات، وهي جزء من البرنامج يمكن إعادة استخدامه. نحن لا نريد أن نعطيك التعريف المحدد والدقيق للكئنات والأصناف، لكن أقترح أن نستخدمهم بشكل مباشر وليس جزئي. سوف نفهم هذا تدريجيا.

   في السطر الثاني من مثالنا : `fen1 = Tk()`، نحن استخدمنا صنفا للوحدة tkinter، والصنف **Tk**()، ونحن أنشأنا مثيل (اسم آخر يصف كائنا محددا)، أي النافذة **fen1**.

   هذه عملية تمثيل كائن من العمليات الأساسية في التقنيات الحالية للبرمجة. هذه الطريقة في الواقع الأكثر استخداما وتعرف باسم البرمجة الشيئية (أو OOP أي البرمجة الموجهة).

---

28 الودجة هي نتيجة لانكماش عبارة نافذة الأداة. في بعض لغات البرمجة , هذه ليست ما يطلق عليها السيطرة أو المكون الرسومي هذا المصطلح يشير إلى أي شئ يمكن وضعه في إطار التطبيق : مثل الزر والصور إلخ... وأحيانا النافذة نفسها.

29 إذا قمت بإجراء هذه العملية تحت نظام ويندوز , يجب عليك استخدام ويفضل أن يكون الإصدار القياسي من بيثون في إطار دوس في بيئة تطوير متكاملة IDLE أو PythonWin بدلا من ذلك. يمكنك أن ترى أفضل , مايحدث بعد إدخال كل أمر .

الصنف هو نموذج عام يبدأ من أن نطلب من الآلة بناء كائن حاسوبي معين. الصنف يحتوي على مجموعة من التعريفات للخيارات المختلفة، نحن لن نستخدم سوى جزء من الكائن الذي صنعناه إبتداءا منها. وبالتالي الصنف **Tk()**، الذي يعد من الفئات الرئيسية لمكتبة tkinter، ويحتوي على كل ما هو مطلوب لتوليد أنواع مختلفة من نوافذ التطبيقات، مختلفة الأحجام والألوان، مع أو بدون شريط أوامر ... إلخ.

نحن نستخدمها هنا لصناعة كائن رسومي أساسي، أي نافذة تحتوي على كل ما تبقى. في أقواس **Tk()**- يمكننا تحديد خيارات مختلفة، لكن سنترك هذا إلى وقت آخر.

تجسيد التعليمة يشبه تعيين بسيط لمتغير. أفهم من ذلك أنه يحدث هنا شيئان في وقت واحد :

- إنشاء كائن جديد، (و الذي قد يكون معقدا للغاية في بعض الحالات، وبالتالي يحتل مساحة كبيرة في الذاكرة)
- تعيين المتغير، والذي سيعمل الآن كمرجع لمعالجة الكائن.[30]

3. في السطر الثالث :

```
tex1 = Label(fen1, text='Bonjour tout le monde !', fg='red'),
```

نحن سنصنع كائنا آخر (ودجة)، وهذه المرة من الصنف **Label()**.

كما يوحي لنا اسمه، هذا الصنف يعرف جميع أنواع التسميات (أو العلامات). في الواقع، هو ببساطة هو جزء من النص، يستخدم لعرض معلومات ورسائل مختلفة داخل النافذة.

سنسعى جاهدين لتمرير الطريقة الصحيحة للتعبير عن الأشياء، نقول هنا أننا صنعنا الكائن **tex1** بواسطة مثيل الصنف **Label()**.

لاحظ أننا قمنا باستدعاء الصنف، بنفس الطريقة التي استدعينا فيها الدالة : وهذا معناه تقديم عدد من البرامترات داخل الأقواس. سوف نرى لاحقا أن الصنف هو نوع من أنواع "الحاويات"، والتي تم تجميع فيها مجموعة من الدالات والمعطيات.

ما هي البرامترات التي قدمناها لهذا المثل ؟

- البرامتر الأول الذي تم تمريره هو (**fen1**)- يشير إلى أن الودجة الأول الذي قمنا بصنعه داخل الودجة السابقة، التي وضعناها هنا مثل "سيده" : الكائن **fen1** هو الودجة السيد للكائن **tex1**. نستطيع أن نقول أن الكائن **tex1** هو ودجة تابعة للكائن **fen1**.

[30] هذا الاختصار في اللغة هو نتيجة لديناميكية الكتابة من المتغيرات السارية في بيثون، تستخدم اللغات الأخرى تعليمة خاصة (نحو `new`)، لإنشاء مثيل كائن جديد. مثال: `maVoiture = new Cadillac` (instanciation d'un objet de classe `Cadillac`، المشار إليها في المتغير `maVoiture`).

- هذان البرامتران يستخدمان ليصفان بالضبط ماذا يجب أن تأخذ الودجة. هذا في الواقع اختياران للصنع، قدم لكل واحد في شكل سلسلة نصية / في البداية نص التسمية، ثم اللون (foreground أو باختصار **fg**). نحن نريد أن يظهر النص بشكل جيد، لذلك لوناه باللون الأحمر.

و يمكننا أيضا تحديد المزيد من الخصائص الأخرى : مثل الخط أو اللون الخلفي على سبيل المثال. كل هذه الخصائص لديها قيم افتراضية في تعريف الصنف **Label()**. لا يمكننا تحديد جميع الخيارات المتاحة للخصائص المختلفة عن النموذج القياسي.

4. في السطر الرابع من مثالنا : `tex1.pack()`ـ فعّلنا الأسلوب المرتبط بالكائن **tex1** : الأسلوب **pack()**. لقد التقينا بالفعل مع هذا الأسلوب (عن القوائم خاصةً). وهنالك أسلوب الدالة مضمنة في الكائن (نقول أيضا كما يتم تغليف الكائن). وسوف نعلم عما قريب أن الكائن الحاسوبي هو في الواقع عنصر لبرنامج يحتوي دائما على :

- عدد من البيانات (رقمية أو غيرها)، تحتوي في داخل المتغيرات من أنواع مختلفة : نسميها خصائص الكائن.
- و يطلق على مجموعة من الإجراءات والدالات (والتي هي خوارزمية) : أساليب الكائن.

الأسلوب **pack()** هو مجموعة من الأساليب التي تطبق ليس فقط على ودجة الصنف **Label()**ـ. بل تطبق في معظم الودجات الأخرى لـ *tkinter*، والتي تؤثر على ترتيبها في الإطار الهندسي-في النافذة. كما يمكنك أن ترى بنفسك إذا قمت بإدخال أوامر مثالنا واحدا تلو الآخر، الأسلوب **pack()** يقلل تلقائيا حجم نافذة - السيد - بحيث تكون كبيرة لإضافة ما يكفي من الويدجات - التابعة - المحددة مسبقا.

5. في السطر الخامس :

```
bou1 = Button(fen1, text='Quitter', command = fen1.destroy),
```

صنعنا الودجة الثانية - "تابع" - : وزر

كما فعلنا مع الودجة السابقة، نحن استدعينا الصنف **Button()** مصحوبا بقوسين بداخلها البرامترات. لأنه في هذه الحالة من الكائن التفاعلي، يجب علينا أن نضع خيار ماذا سيحدث عندما يقوم المستخدم بالضغط على الزر. في هذه الحالة، وضعنا خيار إغلاق مرتبط بالكائن **fen1**، الذي ينبغي أن يتسبب بإغلاق النافذة[31].

6. في السطر السادس استخدمنا الأسلوب **pack()** حتى يتكيف هندسيا في النافذة مع الكائن الجديد لدمجه.

---

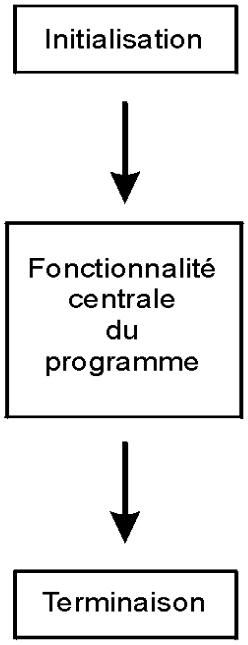
[31] **تحذير : إستدعاء الأسلوب** "destroy" **لا يتم هنا (أي داخل تعليمة وصف الزر) . و لذلك لا يجب إلحاق أقواس بإسمه. لأن** tkinter **هو الذي سيتولى إستدعاء** destroy() **عندما يقوم المستخدم بضغط الزر.**

7. في السطر السابع : `fen1.mainloop()` مهم للغاية، لأنه يتسبب ببدء الأحداث المرتبطة بالنافذة. هذه التعليمة ضرورية للغاية لتطبيقنا سواء لـ - الإطلاع - على نقرات الفأرة، أو للضغطات على لوحة المفاتيح، إلخ ... إذا هذه التعليمة بتعبير آخر - تجعله يعمل -.

كما يوحي اسمها (*mainloop*)، هو أسلوب للكائن **fen1**، الذي يفعّل حلقة البرنامج، الذي يعمل في الخلفية بشكل مستمر، في انتظار رسائل من قبل نظام التشغيل المثبت على الحاسوب. ينتظر في الواقع بشكل مستمر في بيئته، أجهزة الإدخال (الفأرة، لوحة المفاتيح، إلخ ...). عندما يتم الكشف عن أي حالة، يتم إرسال رسائل مختلفة التي تصف الحالة إلى البرنامج. ستتعرف على التفاصيل قريبا.

برامج تتوجه حسب الأحداث

لقد صنعت برنامجك الأول مستخدما الواجهة الرسومية. هذا النوع من البرامج يتنظم بطريقة مختلفة عن السكريبتات التي درسناها سابقا.

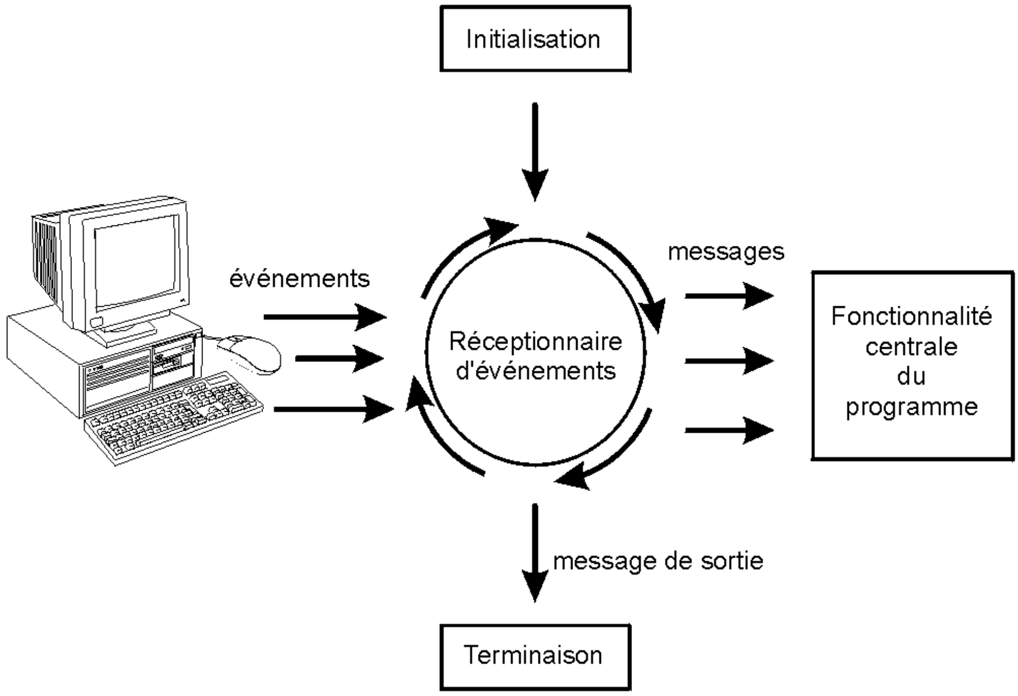


جميع برامج الحاسوب لديك تعمل بثلاثة مراحل رئيسية : مرحلة التهيئة، التي تحتوي على التعليمات التي تعدّ العمل المطلوب (استدعاء الوحدات الخارجية اللازمة،فتح ملفات، الاتصال بخادم قواعد البيانات أو في شبكة الانترنات، إلخ ..)، المرحلة الوسطى (المركزية) حيث نجد هناك الدالات الرئيسية للبرنامج (هذا معناه كل شيء من المفترض أن يفعله البرنامج : عرض البيانات على الشاشة، تنفيذ العمليات الحسابية، تحرير محتويات لملف، طباعة، إلخ ...)، وفي النهاية مرحلة الانتهاء والذي تعمل على إغلاق المعاملات - بشكل صحيح - (هذا معناه إغلاق الملفات المفتوحة، قطع الاتصالات الخارجية، إلخ ...)

في البرنامج - بالوضع النصي - ، يتم ترتيب هذه المراحل الثلاثة ببساطة في نمط خطي كما تم توضيحه. وبناءً على ذلك، تتميز هذه البرامج بتفاعلها المحدود جدا مع المستخدم. هذه من الناحية العملية ليس لديك أي حرية : يطلب منك من وقت لأخر إدخال بعض البيانات من لوحة المفاتيح، لكن دائما في ترتيب محدد سابقا لسلسلة من تعليمات البرنامج.

في حالة أن البرنامج يستخدم الواجهة الرسومية، يكون التنظيم الداخلي هو المختلف. نقول أن البرنامج يتوجه بواسطة الأحداث. بعد مرحلة التهيئة، البرنامج من هذا النوع يبقى ينتظر، ويمرر السيطرة على برنامج آخر، والتي هي أكثر أو أقل اندماجا مع نظام التشغيل الموجود على الحاسوب.

هذا متلقي الأحداث يقوم باستمرار بفحص الملحقات (لوحة المفاتيح، الفأرة، الساعة، المودم، إلخ ...) ويتفاعل فور الكشف عن حصول حدث. ويمكن أن يكون هذا الحدث من المستخدم : تحريك الفأرة، الضغط على مفتاح في لوحة المفاتيح، إلخ ... أو يمكن حدث خارجي أو تلقائي (انتهاء المؤقت، على سبيل المثال) .



عندما يتم كشف حدث، يرسل المتلقي رسالة معينة إلى البرنامج[32]، الذي هو مصمم ليرد وفقا لذلك.

مرحلة التهيئة لبرنامج يستخدم واجهة رسومية تتضمن مجموعة من التعليمات التي تضع مكونات الواجهة التفاعلية في مكانها (النوافذ، الأزرار، الخانات، إلخ ...). المزيد من تعليمات التي تعرف رسائل الأحداث تكون مدعومة : في الواقع، يمكن للمرء أن يقرر ردة فعل البرنامج على أحداث معينة ويتجاهل البقية.

بينما المرحلة الوسطى في البرنامج النصي، تتكون من سلسلة من التعليمات التي تصف ترتيب المهام التي ينبغي أن يؤديها البرنامج (حتى لو تم تقديمها في مسارات مختلفة استجابة للظروف التي تواجهه)، لا توجد مرحلة وسطى في البرنامج الذي يستخدم الواجهة الرسومية بل تكون مجموعة من الدالات المستقلة. وتستدعى كل دالة خاصة عندما يتم الكشف عن حدث معين من قبل نظام التشغيل : يتم تنفيذ الدالة لتقوم بالعمل المتوقع للبرنامج في استجابة لهذا الحدث، ثم لا شيء آخر[33].

32هذه الرسائل غالبا ما تدل على WM (رسائل النافذة) في بيئة رسومية تتكون من نوافذ (مع مناطق فعالة كثيرة ؛ الأزرار، خانات الاختيار، القوائم، إلخ). في وصف الخوارزميات، كما يحدث في كثيرة من الأحيان تختلط هذه الرسائل مع الأحداث نفسها .

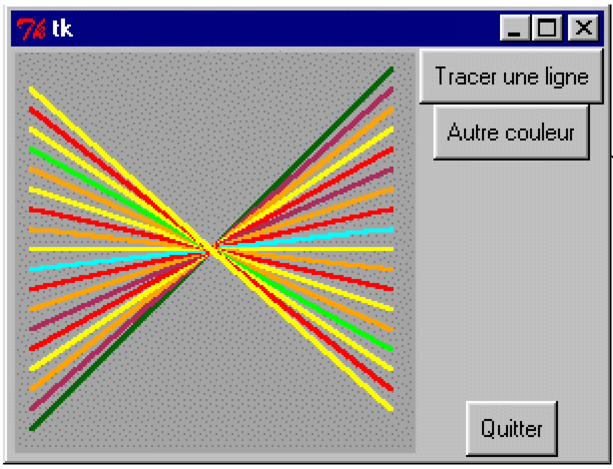
33بالمعنى الدقيق للكلمة، أي دالة لا ترجع أية قيمة هي إجراء (انظر إلى صفحة70).

إذا حَدَثَ حدْث آخر، يمكن أن يكون الرد من الدالة الثانية (أو الثالثة، أو الرابعة، إلخ...)التي سوف يتم تفعيلها لتبدأ عملها بالتوازي مع الدالة الأولى التي لم تكن عملها بعد.[34] يمكن لأنظمة التشغيل ولغات البرمجة الحديثة أن تعمل بالتوازي والتي نسميها أيضا تعدد المهام.

في الفصل السابق، لاحظنا بالفعل أن بنية الملف النصي لبرنامج غير مشابه لبنية الملف عندما يتم تنفيذه. هذه الملاحظة تنطبق أيضا على البرنامج مع الواجهة الرسومية، حيث ترتيب الدالات التي يتم إستدعائها غير مسجلة بأي جزء من البرنامج. الأحداث هي التي تتحكم !

كل هذا قد يبدو معقدا قليلا. سوف نوضح هذا في بعض أمثلة.

مثال رسومي : رسم خطوط على اللوحة



السكريبت الذي بالأسفل يصنع نافذة مع ثلاثة أزرار ولوحة. بمصطلح tkinter، اللوحة - canevas - هي مساحة مستطيلة محددة، يمكن أن يوضع بها مختلف التصاميم والصور باستخدام أساليب محددة.[35]

عند الضغط على زر "رسم خط" - "Tracer une ligne"، سيظهر سطر ملون جديد على اللوحة، كل واحد لديها ميل مختلف عن سابقتها.

إذا تم الضغط على زر "لون آخر" - " Autre couleur "، سيتم اختيار لون جديد من سلسلة الألوان المحددة. هذا اللون سيتم استخدامه في الرسم القادم.

زر "خروج" - " Quitter " لإنهاء التطبيق عن طريق غلق النافذة.

```
# تمرين صغير يستخدم مكتبة الرسومية tkinter

from tkinter import *
from random import randrange

# --- تعريف دالات لمعالجة الأحداث : ---
def drawline():
    "Tracé d'une ligne dans le canevas can1"
    global x1, y1, x2, y2, coul
    can1.create_line(x1,y1,x2,y2,width=2,fill=coul)
```

---

34 نفس الدالة يمكن أن يتم استدعاؤها عدة مرات ردا على وقوع بعض الأحداث نفسها, ثم يتم تنفيذ نفس المهمة بنسخ مختلفة . سوف نرى لاحقا أنه يمكن أن يؤدي إلى "آثار حافة" مزعجة.

35 في النهاية سيتم تحريك هذه الرسوم في مرحلة لاحقة .

```
    # تعديل الإحداثيات للسطر التالي :
    y2, y1 = y2+10, y1-10

def changecolor():
    "Changement aléatoire de la couleur du tracé"
    global coul
    pal=['purple','cyan','maroon','green','red','blue','orange','yellow']
    c = randrange(8)         # => توليد رقم عشوائي بين 0 و 7
    coul = pal[c]

#------ البرنامج الرئيسي -------

# سيتم استخدام المتغيرات التالية بشكل عام :
x1, y1, x2, y2 = 10, 190, 190, 10      # إحداثيات السطر
coul = 'dark green'                   # لون السطر

# إنشاء الودجة الرئيسية ("السيد") :
fen1 = Tk()
# إنشاء الودجة ("التابع") :
can1 = Canvas(fen1,bg='dark grey',height=200,width=200)
can1.pack(side=LEFT)
bou1 = Button(fen1,text='Quitter',command=fen1.quit)
bou1.pack(side=BOTTOM)
bou2 = Button(fen1,text='Tracer une ligne',command=drawline)
bou2.pack()
bou3 = Button(fen1,text='Autre couleur',command=changecolor)
bou3.pack()

fen1.mainloop()   # بدء إستقبال الأحداث

fen1.destroy()    # تدمير (غلق) النافذة
```

وفقا لما شرحناه في صفحات السابقة، وظيفية هذا البرنامج تقوم على دالتين أساسيتين **drawline()** و **changecolor()**، التي يتم تفعيلها من خلال الأحداث، لأنها تم تفعيلها في مرحلة التهيئة.

في هذه المرحلة - مرحلة التهيئة -، نحن نبدأ باستدعاء وحدة tkinter بالإضافة إلى وحدة random التي تقوم باختيار رقم عشوائي. ثم صنعنا الويدجات المختلفة مثل **Canvas**، **Tk()** و **Button()**. لاحظ أن بالتمرير للصنف **Button()** صنعنا مجموعة من الأزرار، التي هي مشابهة لبعضها جدا، مع خيارات لكل واحدة منها لصناعتها، والأزرار تستطيع أن تعمل بشكل مستقل عن الأخرى .

مرحلة التهيئة تنتهي مع التعليمة **fen1.mainloop()** التي تبدأ بتلقي الأحداث. التعليمات التي تأتي بعدها ستعمل عندما يتم الخروج من الحلقة، ستخرج من خلال أسلوب **fen1.quit()** (انظر أدناه).

خيار الأمر المستخدم في عبارة تجسيد الأزرار التي ترسم الدالة التي سيتم استدعائها عندما يعمل هذا الحدث " ضغط على الأزر الأيسر للفأرة على الودجة". في الواقع يجب عمل اختصار لهذا الحدث حاصة، والذي يتم تقديمه من tkinter لراحتك

لأن هذا الحدث يرتبط بشكل طبيعي مع الودجة من نوع زر. سوف نرى لاحقا أن هنالك تقنيات أخرى أكثر عمومية لربط أي نوع من الأحداث إلى أي قطعة.

يمكن للدالات في هذا السكريبت تعديل - تحرير - قيم المتغير المعرفة في الجزء الرئيسي من البرنامج. ولقد أصبح هذا ممكنا مع التعليمة **global** المستخدمة لتعريف هذه الدالات. نحن نسمح لأنفسنا أن نفعل ذلك لبعض الوقت (حتى لو كان فقط للتعود على التمييز بين المتغيرات المحلية والعامة)، لكن كما ستفهم في وقت لاحق، هذه الممارسة غير مستحسنة، خاصة عندما تكتب برامج كبيرة. سوف نتعلم أفضل التقنيات عندما نصل لدراسة الأصناف (بداية من الصفحة 175).

في دالتنا **changecolor()**، يتم اختيار لون عشوائي من القائمة. وللقيام بذلك قمنا باستخدام الدالة **randrange()** التي تقوم باستدعاء الوحدة random. التي يتم استدعاؤها مع البرامتر **N**ـ هذه الدالة تقوم بإرجاع عدد صحيح، ما بين **0** و **N-1**.

زر الأمر مرتبط بـ - Quitter - خروج - التي تستدعي الأسوب **quit()** لنافذة **fen1**. ويستخدم هذا الأسلوب لإغلاق - خروج - من متلقي الأحداث (**mainloop**) المرتبط بهذه النافذة. عندما يتم تفعيل هذا الأسلوب، سيتواصل تنفيذ البرنامج مع التعليمة بعد استدعاء الـ **mainloop**. في مثالنا، سيتم إزالة النافذة .

تمارين

1.8 كيف يتم تغيير البرنامج لكي تكون ألوان الخطوط : cyan و maroon و green؟

2.8 كيف يتم تغيير البرنامج لكي تكون جميع الخطوط أفقية وعمودية ؟

3.8 كبر حجم اللوحة لتصبح عرضها 500 وحدة وارتفاعها 650 وحدة. وغيرـ أيضا حجم الخطوط، لتكون حوافهم تتساوى مع حواف اللوحة.

4.8 أضف دالة **drawli ne2** التي تتبع خطين أحمرين بعلامة أكس في وسط اللوحة، واحدة أفقية والأخرى عمودية. وأضف أيضا زر "منظار"، عند الضغط عليه سوف تظهر علامة أكس.

5.8 كرر كتابة البرنامج الأول. أستبدل الأسلوب **create_line** بـ **create_rectangle**. ماذا سيحدث ؟ و بنفس الطريقة حاول مع **create_oval**، **create_arc** و **create_polygon**.

لكل أسلوب، اكتب خياراته في البرامترات. (ملاحظة : بالنسبة للمضلع، فمن الضروري القيام بتعديل صغير على البرنامج ليعمل !)

6.8 - احذف السطر **global x1, y1, x2, y2** في دالة **drawline** في البرنامج الأصلي. ماذا حدث؟ ولماذا؟
- إذا وضعت "x1, y1, x2, y2" داخل الأقواس، في سطر تعريف الدالة **drawline**ـ بطريقة لتمرير المتغيرات للدالة كبرامترات، هل يعمل البرنامج ؟ لا تنسَ أيضا تغيير السطر الذي يستدعي هذه الدالة !

إذا عرّفت **x1, y1, x2, y2 = 10, 390, 390, 10** بدل **global x1, y1** ...e ماذا سيحدث ؟ ولماذا ؟ ماذا استنتجت من هذا ؟

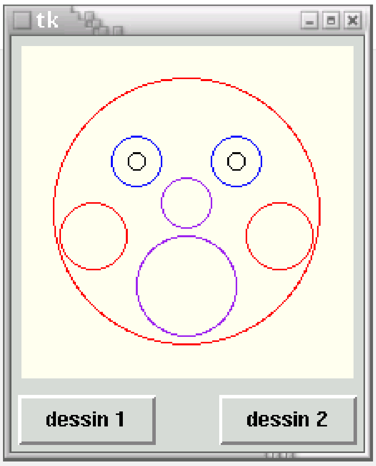
7.8 أ) اكتب برنامجا قصيرا يرسم الحلقات الأولمبية الخمس في مستطيل أبيض (white). بالإضافة إلى زر Quitter للخروج من المافذة.

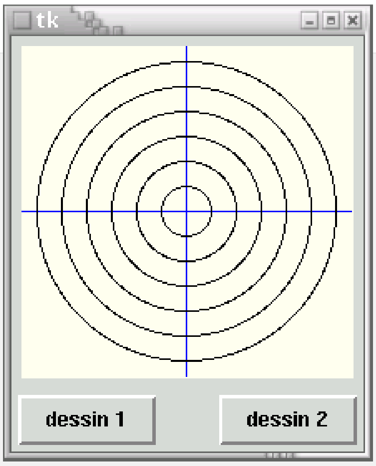
ب) عدل البرنامج أعلاه بإضافة خمسة أزرار كل زر يرسم حلقة من الحلقات الخمس.

8.8 في دفتر الملاحظات، خطط جدول من عمودين. ستكتب على اليسار تعريفات الأصناف التي قد درسناها (مع قائمة البرامترات)، وعلى اليمين الأساليب المرتبطة بهذه الأصناف (مع برامتراتها). اترك بعض المجال لإكماله في وقت لاحق.

مثال رسومي : رسمان متناوبان

المثال التالي يظهر لك كيفية الاستفادة من المعلومات والمعرفة التي قد حصلت عليها للقوائم الحلقات والدالات، لرسم العديد من الرسومات بأسطر قليلة. هذا البرنامج الصغير يظهر واحد من الرسمين الموجودين بالأسفل، على حسب الزر المضغوط :






```
from tkinter import *

def cercle(x, y, r, coul ='black'):
    "tracé d'un cercle de centre (x,y) et de rayon r"
    can.create_oval(x-r, y-r, x+r, y+r, outline=coul)

def figure_1():
    "dessiner une cible"
    # أحذف أولا أي رسم سابق :
    can.delete(ALL)
    # أرسم خطين (أفقي وعمودي) :
    can.create_line(100, 0, 100, 200, fill ='blue')
    can.create_line(0, 100, 200, 100, fill ='blue')
    # أرسم عدة دوائر متحدة المركز :
    rayon = 15
    while rayon < 100:
```

```
        cercle(100, 100, rayon)
        rayon += 15

def figure_2():
    "dessiner un visage simplifié"
    # أحذف أولا أي رسم سابقا :
    can.delete(ALL)
    # خصائص كل دائرة
    # موضوعة في قائمة من القوائم :
    cc =[[100, 100, 80, 'red'],      # الوجه
         [70, 70, 15, 'blue'],       # العينان
         [130, 70, 15, 'blue'],
         [70, 70, 5, 'black'],
         [130, 70, 5, 'black'],
         [44, 115, 20, 'red'],       # الخدان
         [156, 115, 20, 'red'],
         [100, 95, 15, 'purple'],    # الأنف
         [100, 145, 30, 'purple']]   # الفم
    # يتم رسم جميع الدوائر بمساعدة حلقة :
    i =0
    while i < len(cc):       # تدوير الحلقة
        el = cc[i]           # كل عنصر هو في حد ذاته قائمة
        cercle(el[0], el[1], el[2], el[3])
        i += 1

##### البرنامج الرئيسي : ############

fen = Tk()
can = Canvas(fen, width =200, height =200, bg ='ivory')
can.pack(side =TOP, padx =5, pady =5)
b1 = Button(fen, text ='dessin 1', command =figure_1)
b1.pack(side =LEFT, padx =3, pady =3)
b2 = Button(fen, text ='dessin 2', command =figure_2)
b2.pack(side =RIGHT, padx =3, pady =3)
fen.mainloop()
```

ابدأ بتحليل البرنامج الرئيسي، في نهاية السكريبت :

لقد قمنا بإنشاء نافذة، بتمثيل كائن للصنف **Tk()** في المتغير **fen**.

لقد قمنا بإنشاء نافذة، بتمثيل كائن للصنف **Tk()** في المتغير **can**. ثم قمنا بوضع ثلاثة ويدجات في هذه النافذة : لوحة وزران. ولقد أنشأنا اللوحة في المتغير can، الزران في المتغير **b1** و **b2**. كما في السكريبت السابق، الويدجات تم وضعهم في أماكنهم في النافذة بمساعدة الأسلوب **pack()**، لكن هذه المرة استخدمنا هذه الخيارات :

- الخيار **side** الذي يقبل القيم LEFT، BOTTOM، TOP أو RIGHT لوضع الودجة في الجانب المناسب في النافذة. هذه الأسماء تكتب بأحرف كبيرة وهم جزء من متغيرات التي تم استدعاؤها مع الوحدة tkinter، ويمكن أن تعتبره ك "شبه ثوابت ".

- الخياران **padx** و **pady** اللذان يقومان بحجز مساحة صغيرة حول الودجة. هذه المساحة تعبر عن عدد البيكسلات : **padx** تحجز المساحة على يمين ويسار الودجة، و **pady** تقوم بحجز المساحة فوق وتحت الودجة.
- الأزرار تتحكم في إظهار الرسمين، باستدعاء الدالتين **figure_1()** و **figure_2()**. بما أننا سنرسم العديد من الدوائر في هذه الرسومات، ففكرنا أنه من المفيد أن نبدأ بتعريف دالة **cercle()** لرسم الدوائر. في الحقيقة، وربما لم تكن تعرف سابقا أن اللوحة في tkinter لديها أسلوب **create_oval()** تستطيع من خلاله رسم أية أشكال بيضوية (و بالطبع دوائر أيضا)، لكن هذا الأسلوب يجب أن يحتوي عل أربعة برامترات التي هي إحداثيات أعلى وأسفل ويمين ويسار مستطيل وهمي، في أي شكل بيضوي تريد أن ترسمه. وهذا ليس عمليا في حالات معينة من الدائرة : والأكثر طبيعية هو أن يتم تمرير طول المركز ونصف قطر الدائرة وهذا ما سنحصل عليه في دالتنا **cercle()**، والتي تستدعي الأسلوب **create_oval()** عن طريق إجراء تحويل للتنسيق. لاحظ أيضا أن هذه الدالة يجب إعطاؤها لون الدائرة التي تريدها (اللون الافتراضي هو الأسود).

و عمل هذه الطريقة سهل وواضح في الدالة **figure_1()**ـ لقد صنعنا دالة بسيطة لتكرار رسم جميع الدوائر (بنفس المركز وبنفس القطر متزايد). ملاحظة أخرى وهي استخدام العامل **+=** التي تزيد قيمة المتغير (في مثالنا، المتغيرـ **r** يزداد 15 قيمة في كل تكرار).

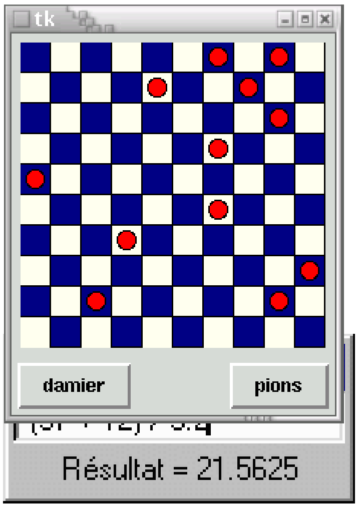
الرسم الثاني هو أكثر تعقيدا نوعا ما، لأنه يتكون من دوائر مختلفة الأحجام في أماكن مختلفة. نستطيع رسم كل هذه الدوائر بمساعدة حلقة تكرار واحدة، إذا عرفنا كيفية الاستفادة من القوائم.

في الحقيقة أن الفرق بين الدوائر التي رسمناها يتلخص في ثلاثة خصائص :

إحداثيات **x** و **y** للوسط، والمركز واللون. لكن دائرة، نستطيع أن نضع هذه الخصائص في قائمة صغيرة، ثم نقوم بجمع كل هذه القوائم الضغيرة في قائمة أكبر. هذا سيتيح لنا قائمة من القوائم، وبمساعدة حلقة التكرار سيتم رسم الدوائر في الأماكن الصحيحة .

تمارين

9.8 ستوحي بعض الأفكار من السكريبت السابق لكتابة برنامج لإظهار لوحة لعبة الداما (الرسم يتكون من مربعات سوداء وبيضاء) عندما نضغط على الزر



10.8 من برنامج التمرين السابق، أضف زرًا سوف يظهر بيادق بشكل عشوائي في لوحة الداما (كل ضغطة على الزر سوف تظهر بيدق بشكل عشوائي).

مثال رسومي : آلة حاسبة بسيطة

على الرغم من أن الكود قصير للغاية، السكريبت الذي بالأسفل مثل الة حاسبة كاملة أي أنه يمكنك الحساب حتى مع الأقواس والرموز العلمية. لا يوجد أي شيء غير عادي. كل هذه الوظائف تستخدم مفسر بدل من مترجم لتنفيذ برامجك.

كما تعلم، فإن المترجم لا يأتي إلا لمرة واحدة، لتحويل الكود المصدري لبرنامجك إلى برنامج قابل للتنفيذ. أي أن دوره ينتهي قبل تنفيذ البرنامج. أما المفسر يبقى نشطا خلال تنفيذ البرنامج وبالتالي هو متوفر لترجمة أي كود مصدري جديد، مثل عبارة رياضية يتم إدخالها عن طريق لوحة المفاتيح من المستخدم.

أي أن لغات البرمجة التي تعتمد على المفسر، يكون مفسرها موجود دائما ليفحص سلسلة نصية مثل تعليمة من اللغة نفسها. يصبح من الممكن بناء بضعة أسطر برمجية من الهيكل البرامج ديناميكي للغاية. في المثال بالأسفل، نحن استخدمنا الدالة **eval()** لفحص التعبير الرياضي الذي تم إدخاله من قبل المستخدم، ثم نظهر نحن نتيجة .

```
# math ووحدة tkinter تمرين يستخدم مكتبة رسومية

from tkinter import *
from math import *

# تعريف الحركة التي يجب اتخاذها عندما يفعله المستخدم
# : مفتاح الإدخال عة الذي يقوم بتحرير حقل الإدخال

def evaluer(event):
    chaine.configure(text = "Résultat = " + str(eval(entree.get())))

# ----- البرنامج الرئيسي : -----

fenetre = Tk()
entree = Entry(fenetre)
entree.bind("<Return>", evaluer)
chaine = Label(fenetre)
entree.pack()
chaine.pack()

fenetre.mainloop()
```

في بداية السكريبت، بدأنا باستدعاء الوحدتين **tkinter** و **math**. هذا الأخير ضروري في الآلة الحاسبة من أجل توفير جميع الدالات الرياضية والعلمية المعتادة : الجيب، جيب التمام، الجذر التربيعي، إلخ.

بعد ذلك نحن عرفنا الدالة **evaluer()**، وهو في الواقع أمر ينفذه البرنامج بعد أن يضغط المستخدم على زر الإدخال بعد أن يدخل التعليمة الرياضية في حقل الإدخال.

هذه الدالة تستخدم الأسلوب **configure()** لودجة **chaine**[36] لتعديل سمة النص. السمة في السؤال يحصل إذا على قيمة جديدة، المحدد بما كتبناه على يمين علامة المساواة : موجود بها سلسلة نصية بنيت بشكل حيوي، بمساعدة دالتين مدمجتين في بيثون : **eval()** و **str()**، ومرتبطة بودجة لـ tkinter : الأسلوب **get()**.

يستخدم **eval()** لفحص تعبير بيثون الممرر إليه في سلسلة نصية ، ناتج هذا الفحص في شكل "رجوع" (إرجاع قيمة). على سبيل المثال :

```
chaine = "(25 + 8)/3"     # سلسلة تحتوي على تعبير رياضي
res = eval(chaine)        # تقييم التعبير الموجود في السلسلة
print(res +5)             # => محتوى المتغير res رقمي
```

يستخدم **str()** لتحويل تعبير رقمي إلى سلسلة نصية. لقد قمنا باستدعاء هذه الدالة لأن العائد السابق يقوم بإرجاع قيمة رقمية، ويجب علينا تحويلها إلى سلسلة نصية لنكون قادرين على دمجها مع رسالة **Résultat =**.

الأسلوب **get()** يرتبط مع الويدجات للصنف **Entry**. في برنامجنا الصغير (مثال)، نحن استخدمنا الودجة من هذا النوع للسماح للمستخدم بإدخال أي عبارة رقمية بمساعدة لوحة مفاتيحه ويقوم الأسلوب **get()** بأخذ مدخلات المستخدم في الودجة.

نص البرنامج الرئيسي يحتوي على مرحلة التهيئة، التي تنتهي مع مستقبل الأحداث (**mainloop**). ويوجد مثيل للنافذة **Tk()**ـ تحتوي على الودجات **chaine** لتمثيل بداية من الصنف **Label()**، والودجة تدخل مثيل بداية من الصنف **Entry()**.

تحذير : الودجة الأخيرة تقوم حقا بعملها، وهذا معناه نقل التعبير الذي أدخله المستخدم إلى البرنامج، ونحن ربطناه مع الحدث بمساعدة الأسلوب **bind()**[37] :

```
entree.bind("<Return>",evaluer)
```

هذه التعليمة تعني : ربط الحدث - الضغط على زر الإدخال - مع الكائن - الإدخال -، وتعالجها الدالة evaluer.

36 يمكن تطبيق الأسلوب **configure()** لأي ودجة موجودة لتغيير خصائصها

37 كلمة bind معناها ربط.

تم وصف الحدث في سلسلة نصية معينة (في مثالنا، يقصد السلسلة "<Return>". ويوجد عدد كبير من هذه الأحداث (تحريك ونقرات الفأرة، ضغطات على لوحة المفاتيح، تحديد مواقع، وتغيير حجم النوافذ ...إلخ). سوف تجدون قائمة السلاسل المحددة لكل هذه الأحداث في مراجع مكتبة tkinter.

لاحظ جيدا أنه لا يوجد أقواس بعد اسم الدالة **evaluer**. في الحقيقة : في هذه التعليمات، نحن لا نريد استدعاء الدالة (هذا سيكون سابق لأوانه)، ما نريده هو ربط نوع معين من الأحداث مع هذه الوظيفة، بطريقة لنستدعيها في وقت لاحقا، كلما يقع الحدث، إذا وضعنا داخل القوسين، البرامتر الذي سيتم تمريره لأسلوب **bind()** سيكون قيمة رجوع هذه الدالة وليس مرجعها.

سنغتنم هذه الفرصة لندرس عن تركيب التعليمات لتنفيذ أسلوب مرتبط بكائن :

كائن.الأسلوب (البرامترات)

نكتب أولا اسم الكائن الذي نريده ثم نقطة (التي تعمل بمثابة عامل)، ثم اسم الأسلوب الذي نريد تنفيذه. ونضع في ما بين القوسين البرامترا التي تريد تمريرها .

مثال رسومي : كشف وتحديد مكان ضغطة زر الفأرة

في تعريف الدالة evaluer في المثال السابق، ربما لاحظت أننا قد مررنا برامتر الحدث (في ما بين القوسين).

هذا البرامتر إلزامي[38]. عند تعريف دالة لمعالجة الأحداث مرتبطة بأي ودجة بمساعدة الأسلوب **bind()**، ويجب عليك استخدامه دائما كأنه البرامتر الأول. هذه البرامترات في الحقيقة كائن تم صنعها تلقائيا من قبل tkinter، وهو ينقل لمعالج الأحداث عدد من سمات الحدث :

- نوع الحدث : تحريك الفأرة، الضغط على أحد أزرارها، الضغط على زر من لوحة المفاتيح، وضع المؤشر في مكان محدد، فتح أو إغلاق نافذة، إلخ ...
- مجموعة من خصائص الحدث : لحظة وقعها، وخصائص الودجة أو الودجات إلخ ...

لن ندخل في تفاصيل أكثر من ذلك، إذا جربت السكريبت الذي بالأسفل، سوف تفهم بسرعة الفكرة.

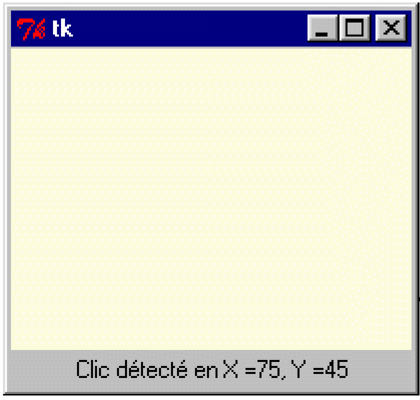
```
# كشف وتحديد موقع نقرة الفأرة في النافذة
from tkinter import *
def pointeur(event):
```

[38] La présence d'un argument est obligatoire, mais le nom *event* est une simple convention. Vous pourriez utiliser un autre nom quelconque à sa place, bien que cela ne soit pas recommandé.

```
    chaine.configure(text = "Clic détecté en X =" + str(event.x) +\
                            ", Y =" + str(event.y))

fen = Tk()
cadre = Frame(fen, width =200, height =150, bg="light yellow")
cadre.bind("<Button-1>", pointeur)
cadre.pack()
chaine = Label(fen)
chaine.pack()

fen.mainloop()
```



السكريبت يعرض نافذة تحتوي على لوحة (**Frame**) صفراء مستطيلة الشكل، وتطلب من المستخدم النقر عليها.

الأسلوب d **bind()** للودجة من نوع لوحة ويرتبط الحدث "الضغط بالزر الأول للفأرة" بمعالج الحدث "المؤشر".

هذا معالج الأحداث يستطيع استخدام السمات **x** و **y** للكائن **event** الذي أنشئ تلقائيا بواسطة tkinter ، ثم لصنع سلسلة نصية التي تعرض موقع الفأرة في لحظة النقر .

تمرين

11.8 عدل البرنامج النصي في الأعلى لإظهار دائرة حمراء صغيرة في المكان الذي نقر عليه المستخدم (سوف تستبدل أول الودجة **Frame** بودجة Canvas).

أصناف الودجة tkinter

> في هذا الكتاب، سوف نعرض لكم تدريجيا استخدام عدد من الويدجات. ولكن ليس في نيتنا تقديم دليل مرجعي كامل ل *tkinter*. ونحن نتقيد هنا فقط بتفسير الويدجات التي تبدو الأكثر إثارة للاهتمام للتعلم الشخصي، وهذا معناه أن نسلط الضوء على المفاهيم البرمجية المهمة، مثل الصنف والكائن. ويرجى الرجوع إلى (13) إذا أردت المزيد من التفصيل .

هنالك 15 صنف أساسي للودجة tkinter :

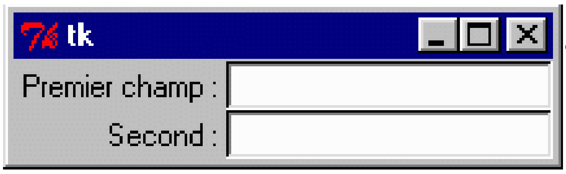
الودجة	الوصف
Button	زر تقليدي، يستخدم لتنفيذ أي أمر .
Canvas	مساحة لوضع مختلف العناصر الرسومية. هذا الودجة تستطيع استخدامها للرسم، صنع وتعديل على رسوم، وأيضا لتطبيق ويدجات خاصة .

الوصف	الودجة
خانة لاختيار أحد الاختيارين (من تحديد المربع أو لا) عند الضغط على هذا الودجة سيتغير الاختيار .	Checkbutton
حقل للمدخلات، والذي من خلال يستطيع المستخدم أن يدخل للبرنامج أي نص من لوحة المفاتيح .	Entry
سطح مستطيل الشكل في النافذة، أين نضع الويدجات الأخرى. هذا السطح يمكن تلوينه. ويمكن أيضا أن تضع له إطار (تزين حوافه).	Frame
أي نص (أو حتى صورة) .	Label
قائمة اختيارات تقدم للمستخدم، عادة ما تقدم في نوع من العلب. ويمكننا ضبطه بطريقة بحيث يصبح مثل "أزرار راديو" أو خانات .	Listbox
قائمة. قد تكون قائمة منسدلة يتم وضعها في شريط العنوان، أو في قائمة "منبثقة" - "pop up" وهي تظهر في أي مكان بعد الضغط بزر الفأرة .	Menu
فزر للقائمة، تستخدم بتشغيل قائمة منسدلة .	Menubutton
تعرض نصا. هذا الودجة هو بديل ودجة ملّصق(Label)، يتكيف تلقائيا حسب النص المعروض إلى حجم معين أو إلى عرض\ارتفاع معين .	Message
يعّرف أنه (نقطة سوداء في دائرة صغيرة) واحد من القيّم لمتغير قد يمتلك أكثر من قيمة. عندما تضغط على زر الراديو يمرر قيمة الزر إلى المتغير، ويمرر فارغ لجميع الأزرار الأخرى لنفس المتغير .	Radiobutton
يسمح لك بتغيير قيمة متغير بطريقة مرئية عن طريق تحريك المؤشر على طول المسطرة .	Scale
تستطيع استخدامه مع العديد من الحاجيات : لوحة، نص، قائمة مربعات ...إلخ	Scrollbar
يستخدم لعرض نص منسق. كما يسمح للمستخدم تعدل (تحرير) النص المعروض. ويمكن إدراج صور أيضا .	Text
نافذة عرض منفصلة، تظهر عند البداية .	Toplevel

هذه الأصناف لويدجات تحتوى كل واحد منهم عدد كبير من الأساليب. ويمكننا أيضا ربطها بالأحداث، كما رأينا في الصفحات السابقة. وسوف نتعلم كيفية وضع كل هذه الحاجيات في النوافذ باستخدام ثلاثة أساليب مختلفة : الأسلوب **grid**()، والأسلوب **pack**() والأسلوب **place**().

الفائدة من استخدام هذه الأساليب هو أن نجعل هذه البرامج محمولة (وهذا معناه أن تعمل جيدا في جميع أنظمة التشغيل المختلفة مثل يونكس أو ماك أو ويندوز)، ويمكن تغيير حجم نوافذها .

استخدام الأسلوب grid للتحكم في أماكن الويدجات



حتى الآن، نحن نستخدم دائما الأسلوب **pack()**. لوضع الويدجات على النوافذ. هذا الأسلوب يتميز بأنه بسيط للغاية، لكن لا يعطينا الحرية الكاملة في وضع الويدجات كما يحلو لنا. ماذا نفعل، على سبيل المثال، للحصول على نافذة مثل التي بالأسفل ؟

يمكننا أن نقوم بعدد من المحاولات لتجربة استخدام الأسلوب **pack()** مع البرامترات نوع "side="، مثل التي قمنا بفعلها سابقا، لكن هذه الطريقة لا تفيدنا كثيرا. مثلا لو كتبت :

```
from tkinter import *

fen1 = Tk()
txt1 = Label(fen1, text = 'Premier champ :')
txt2 = Label(fen1, text = 'Second :')
entr1 = Entry(fen1)
entr2 = Entry(fen1)
txt1.pack(side =LEFT)
txt2.pack(side =LEFT)
entr1.pack(side =RIGHT)
entr2.pack(side =RIGHT)

fen1.mainloop()
```

ستكون النتيجة ليس ما كنا نريده !



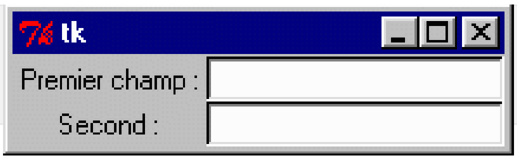
لتفهم بشكل أفضل الأسلوب **pack()**، يمكنك تجربة العديد من الخيارات، مثل **side =BOTTOM**، **side =TOP**، لكل واحدة من الويدجات الأربعة. لكنك بالتأكيد لن تحصل على ما أردناه هنا. يمكنك فعل هذا من خلال تعريف ودجتان من نوع إطار **Frame()** s إضافية، ثم ستقوم بدمج الويدجات **Label()** و **Entry()**. وسيكون هذا معقدا للغاية.

لقد حان الوقت لتعلم استخدام أسلوب جديد لحل هذه المشكلة. يرجى منك الآن تحليل السكريبت بالأسفل : يحتوي (تقريبا) على الحل :

```
from tkinter import *

fen1 = Tk()
txt1 = Label(fen1, text = 'Premier champ :')
txt2 = Label(fen1, text = 'Second :')
entr1 = Entry(fen1)
entr2 = Entry(fen1)
txt1.grid(row =0)
```

```
txt2.grid(row =1)
entr1.grid(row =0, column =1)
entr2.grid(row =1, column =1)
fen1.mainloop()
```



في هذا السكريبت، لقد قمنا باستبدال الأسلوب **pack()** بالأسلوب **grid()**. كما ترون، إن استخدام الأسلوب **grid()** بسيط للغاية. هذا الأسلوب يعتبر النافذة كأنها جدول (أو شبكة). ثم ستكتفي أنت بالإشارة إلى الصف والعمود من الجدول الذي تريد وضع به الودجة. يمكنك ترقيم الأعمدة والصفوف كما تريد ، ابتداء من أي رقم، مثلا 0 أو 1 أو 2 ...إلخ : يقوم tkinter بتجاهل الصفوف والأعمدة الفارغة. إذا لم تضع أي رقم لسطر أو لعمود، ستكون القيمة الافتراضية 0.

يقوم tkinter تلقائيا بتحديد عدد الصفوف والأعمدة اللازمة. ولكن ليس هذا فقط : فإذا حللت النافذة الصغيرة الذي أنتجها السكريبت الذي بالأعلى، ستجد أننا لم نصل لهدفنا بعد. السلسلتان في الجزء الأيسر من النافذة موجودتان في الوسط، إذا يجب علينا أن نجعلها على اليمين. لتحقيق هذا، يكفي أن نضيف برامتر عند استدعاء الأسلوب **grid()** لهذه الويدجات. الخيار **sticky** يمكن أن يأخذ واحدا من هذه القيم الأربعة : **N, S, W, E** (الاتجاهات الأربعة باللغة الأنكليزية). على أساس هذه القيم، ستكون محاذات الويدجات أعلى أو أسفل، أو على اليمين أو على اليسار. سنستبدل إذا السطرين الأولين لتعليمات **grid()** في السكريبت بـ :

```
txt1.grid(row =0, sticky =E)
txt2.grid(row =1, sticky =E)
```

و أخيرا سنحصل على ما نريده.

حلل الآن النافذة التالية :



هذه النافذة تتكون من ثلاثة أعمدة : الأولى تتكون من 3 سلاسلة نصية، الثانية تتكون من 3 حقول للإدخال، وأما الثالثة فتتكون من صورة. أول عمودين يتكونان من ثلاثة صفوف، لكن الصورة التي تقع في العمود الثالث تنتشر على ثلاثة صفوف.

كود هذه النافذة :

```
from tkinter import *

fen1 = Tk()

# صنع ودجتي 'Label' و'Entry' :
txt1 = Label(fen1, text ='Premier champ :')
txt2 = Label(fen1, text ='Second :')
txt3 = Label(fen1, text ='Troisième :')
entr1 = Entry(fen1)
entr2 = Entry(fen1)
entr3 = Entry(fen1)

# صنع الودجة 'لوحة' تحتوي على صورة نقطية :
can1 = Canvas(fen1, width =160, height =160, bg ='white')
photo = PhotoImage(file ='martin_p.gif')
item = can1.create_image(80, 80, image =photo)

# تنسيق باستخدام الأسلوب "grid" :
txt1.grid(row =1, sticky =E)
txt2.grid(row =2, sticky =E)
txt3.grid(row =3, sticky =E)
entr1.grid(row =1, column =2)
entr2.grid(row =2, column =2)
entr3.grid(row =3, column =2)
can1.grid(row =1, column =3, rowspan =3, padx =10, pady =5)

# بدء التشغيل :
fen1.mainloop()
```

لتشغيل هذا السكريبت، يجب عليك أولا أن تغير اسم ملف الصورة martin_p.gif باسم الصورة التي تريدها ويجب أن تكون بنفس مكان السكريبت. انتبه : مكتبة tkinter القياسية لا تقبل سوى عدد قليل من أنواع الصور. من هذه الأنواع GIF[39].

1.يمكننا أن نلاحظ بعض الأشياء في السكريبت :1. التقنية المستخدمة لتضمين الصورة :

إن tkinter لا تسمح لك بتضمين الصور مباشرة في النافذة. يجب عليك أولا وضع لوحة (canevas)، ثم نضع الصورة في اللوحة. نحن اخترنا اللوحة بلون أبيض، لكي نميزها عن النافذة. يمكنك استبدال البرامتر `bg ='white'` بـ `bg ='gray'` إذا أردت أن تصبح اللوحة غير مرئية. بما أنه يوجد العديد من أنواع الصور، يجب عينا أن نعرف كائن الصورة على أنه صورة نقطية GIF، وذلك عن طريق الصنف **PhotoImage()**.

39الأنواع الأخرى من الرسوم ممكنة، لكنها تتطلب وحدات رسومية لمكتبة PIL (Python Imaging Library) والتي هي امتداد لبيثون متاحة على : http://www.pythonware.com/products/pil/. وهذه المكتبة تسمح لك بأداء العديد من المعالجات على الصور. ولكن دراسة هذه التقنيات خارج إطار دورتنا.

2.سطر الذي وضعنا الصورة في اللوحة :

```
item = can1.create_image(80, 80, image =photo)
```

لاستخدام طريقة صحيحة، نحن ننصح هنا باستخدام الأسلوب **create_image()** مرتبطة بالكائن **can1** (و الذي هو كائن مثيل للصنف "لوحة" (**Canvas**). أول برامترين يمرران (80، 80) وهي إحداثيات **x** و **y** للوحة حيث يتم وضعها في المنتصف. إن أبعاد اللوحة هي 160 × 160، وإن اختيارنا سيؤدي إلى وضع الصورة في منتصف اللوحة.

3.طريقة ترقيم الصفوف والأعمدة في أسلوب **grid()** :

يمكنك أن ترى أن ترقيم الأعمدة والصفوف في أسلوب **grid()** يبدأ من الرقم 1 (وليس 0 كما في السكريبت السابق). كما قلنا سابقا أن الترقيم حر (أي تبدأ بأي رقم تريده).

يمكننا اختيار أي رقم مثلا : 5،10،15،20... لأن tkinter يقوم بتجاهل كل الصفوف والأعمدة الفارغة. الترقيم من الرقم 1 سيزيد من سهولة قراءة الكود.

4.البرامترات المستخدمة مع **grid()** لوضع اللوحة canevas :

```
can1.grid(row =1, column =3, rowspan =3, padx =10, pady =5)
```

أول برامترين تشير إلى أن اللوحة (Canvas) سوف يتم وضعها في الصف الأول للعامود الثالث. والبرامتر الثالث (**rowspan =3**) يشير إلى أنه سيتم نشره على ثلاثة صفوف. وأما عن (**padx =10, pady =5**) تشير إلى أبعاد الفراغ حول الودجة (الطول والعرض).

5.بما أننا كتبنا الكود واستفدنا من هذا السكريبت كمثال، سوف نقوم الآن بتبسيطه قليلا

تركيب تعليمات لكتابة كود أكثر إيجازا

بيثون من لغات البرمجة عالية المستوى، غالبا يكون من الممكن (و مرغوب فيه) إعادة صياغة السكريبت لجعله أكثر إيجازا.

الكود سيصبح أكثر بساطة، وسيكون في الغالب أكثر قابلية للقراءة. على سبيل المثال تستطيع استبدال السطرين التاليين من السكريبت السابق :

```
txt1 = Label(fen1, text ='Premier champ :')
txt1.grid(row =1, sticky =E)
```

بسطر واحد :

```
Label(fen1, text ='Premier champ :').grid(row =1, sticky =E)
```

في هذا السطر الجديد، يمكنك أن ترى أننا اختصرنا كتابة المتغير **txt1**. ولقد وضعنا المتغير لكي نعيد استخدامه في أماكن أخرى، ولكن هذا ليس ضروريا، ببساطة لقد قمنا باستدعاء الصنف **Label()** الذي يؤدي إلى صنع مثيل لكائن من هذا الصنف. حتى لو لم نخزن مرجع هذا الكائن في متغير (tkinter يحفظها على أية حال في التمثيل الداخلي للنافذة). فإذا تم ذلك، يتم فقدان المرجع لبقية سكريبت بيثون، لكن يمكن أن يتم نقله إلى أسلوب مثل **grid()** في لحظة تمثيله، في عبارة مركبة واحدة. سوف نرى ذلك بأكثر تفاصيل.

حتى الآن، أنشأنا العديد من الكائنات (بواسطة تمثيل بداية من أي صنف)، والتي عيناها في كل مرة إلى المتغيرات. على سبيل المثال، عندما نكتب :

```
txt1 = Label(fen1, text ='Premier champ :')
```

نكون قد صنعنا مثيل من صنف **Label()**، ولقد عيناها إلى المتغير **txt1**.

و يمكن بعد ذلك استخدام المتغير **txt1** للإشارة (مرجع) لهذا المثيل، في مكان آخر لهذا السكريبت، لكننا في الحقيقة لا نستخدمه سوى لمرة واحدة عندما نطبق الأسلوب **grid()**، الودجة هي ليست سوى ملصق(Label) بسيط للوصف. وصنع متغير جديد ليكون مرجعا لمرة واحدة فقط (و مباشرةً بعد إنشائه) ليست فكرة جيدة، لأنه سيحجز بعض المساحة بدون داع.

عندما تكون في هذه الحالة، فمن الأفضل أن تستخدم تعليمات التركيب. على سبيل المثال، يفضل في معظم الأحيان استبدال التعليمتين التاليتين :

```
somme = 45 + 72
print (somme)
```

بتعليمة واحدة مركبة :

```
print (45 + 72)
```

و بالتالي وفرنا متغير.

و بنفس الطريقة، عندما نضع ويدجات لا نرغب في استخدامها في وقت لاحق، كما في الويدجات من صنف **Label()**، والتي يمكنك أن تطبق عليها أسلوب (**pack**، **grid()** أو **place()**) نقوم مباشرة بصنع الودجة في تعليمة مركبة واحدة.

يطبق هذا فقط للويدجات التي لن نشير (كمرجع) إليها بعد صنعها. ويجب على الباقي أن يتم تعيينهم إلى متغير حتى نتمكن من التفاعل معهم في أماكن مختلف في السكريبت.

و في هذه الحالة، نحن ملزمين أن نستخدم تعليمتين منفصلتين، واحدة لتمثيل الودجة، والأخرى لتطبيق الأسلوب عليها. فلا يمكنك، على سبيل المثال، كتابة هذه التعليمة المركبة :

```
entree = Entry(fen1).pack()          # خطأ برمجي !!!
```

و يجب في هذه الحالة، أن نقوم بتمثيل الودجة الجديد وإسناد ذلك إلى المتغير **entree**، ثم سوف يتم تخطيط الصفحة بمساعدة الأسلوب **pack()**.

في الحقيقة، هذه العبارة تولد ودجة جديد من صنف **Entry()**، وبأسلوب **pack()** التي تضعها في النافذة، لكن القيمة التي تم تخزينها في المتغير **entree** ليست مرجعا للودجة ! بل هو قيمة رجوع للأسلوب **pack()** : إذا كنت تاستخدام الأسلوب grid للتحكم في أماكن الويدجات تذكر أن الأساليب مثل الدالات، ترجع دائما قيمة للبرنامج الذي استدعاها. وأنت لا تستطيع أن تفعل شيئا مع قيمة الرجوع : لذا فهو في هذه الحالة كائن فارغ (لاشيء).

و إذا أردت أن تحصل على مرجع حقيقي. يجب عليك استخدام هاتين التعليمتين :

```
entree = Entry(fen1)          # تمثيل الودجة
entree.pack()                 # تطبيق التخطيط
```

> عندما تستخدم الأسلوب ***grid()***. يمكنك ببساطة تبسيط التعليمات البرمجية قليلا. بحذف الإشارة إلى العديد من أرقام الصفوف والأعمدة. من اللحظة التي تستخدم فيها الأسلوب ***grid()*** لوضع الويدجات. سيقوم *tkinter* بوضع الصفوف والأعمدة الضرورية[40] . فإذا كان رقم صف أو عمود غير موجود ، سيتم وضع الودجة في أول مربع فارغ متاح .

السكريبت الذي بالأسفل بدون التبسيط الذي شرحناه :

```
from tkinter import *
fen1 = Tk()

# صنع ويدجات Label(), Entry(), وCheckbutton() :
Label(fen1, text = 'Premier champ :').grid(sticky =E)
Label(fen1, text = 'Deuxième :').grid(sticky =E)
Label(fen1, text = 'Troisième :').grid(sticky =E)
entr1 = Entry(fen1)
entr2 = Entry(fen1)               # من المؤكد أن يتم استخدام مرجع هذه الويدجات لاحقا
entr3 = Entry(fen1)
entr1.grid(row =0, column =1)     # لذلك يجب أن يتم تعيين كل واحدة في متغير مستقل
entr2.grid(row =1, column =1)
entr3.grid(row =2, column =1)
chek1 = Checkbutton(fen1, text ='Case à cocher, pour voir')
chek1.grid(columnspan =2)

# صنع ودجة "لوحة" يحتوي على صورة نقطية :
can1 = Canvas(fen1, width =160, height =160, bg ='white')
photo = PhotoImage(file ='Martin_P.gif')
can1.create_image(80,80, image =photo)
can1.grid(row =0, column =2, rowspan =4, padx =10, pady =5)
```

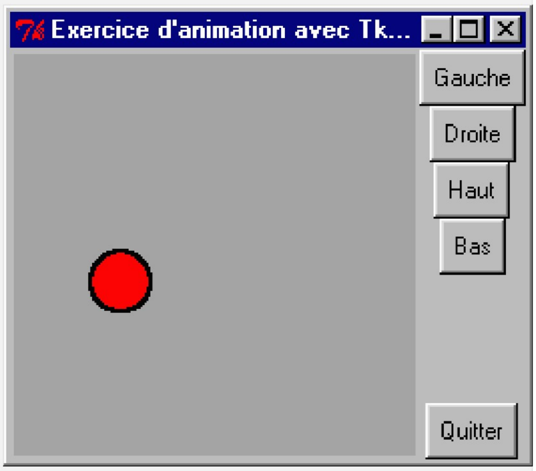
40 لا تستخدم عدة أساليب لتحديد أماكن متعددة في نفس النافذة : الأساليب pack, grid(), و place() لا يمكن جمعها .

```
# بدء التشغيل :
fen1.mainloop()
```

تغيير (تحرير) خصائص كائن - الرسوم المتحركة

في هذه المرحلة من التعليم، ربما تريد أن تظهر رسما صغيرا في اللوحة، ثم يقوم بالتحرك، على سبيل المثال بمساعدة الأزرار.

يجب عليك إذا كتابة، وتجربة وتحليل السكريبت الذي بالأسفل :




```
from tkinter import *

# الإجراء العام للحركة  :
def avance(gd, hb):
    global x1, y1
    x1, y1 = x1 +gd, y1 +hb
    can1.coords(oval1, x1,y1, x1+30,y1+30)

# معالج الأحداث :
def depl_gauche():
    avance(-10, 0)

def depl_droite():
    avance(10, 0)

def depl_haut():
    avance(0, -10)

def depl_bas():
    avance(0, 10)

#------ البرنامج الرئيسي -------

# المتغيرات التالية سيتم استخدامها كمتغيرات عامة  :
x1, y1 = 10, 10   # coordonnées initiales

# صنع الودجة الرئيسية (السد)  :
fen1 = Tk()
fen1.title("Exercice d'animation avec tkinter")

# صنع ويدجات «التابع»  :
can1 = Canvas(fen1,bg='dark grey',height=300,width=300)
oval1 = can1.create_oval(x1,y1,x1+30,y1+30,width=2,fill='red')
can1.pack(side=LEFT)
Button(fen1,text='Quitter',command=fen1.quit).pack(side=BOTTOM)
Button(fen1,text='Gauche',command=depl_gauche).pack()
Button(fen1,text='Droite',command=depl_droite).pack()
Button(fen1,text='Haut',command=depl_haut).pack()
Button(fen1,text='Bas',command=depl_bas).pack()

# بدء متلقي الأحداث (حلقة الأساسية)  :
fen1.mainloop()
```

جسم(محتوى) هذا البرنامج يحتوي على العديد من العناصر التي عرفناها : لقد قمنا بإنشاء النافذة **fen1**، ولقد صنعنا في داخلها لوحة تحتوي على دائرة ملونة، بالإضافة إلى 5 أزرار للتحكم. لاحظ أننا لم نصنع مثيل للويدجات من نوع أزرار في متغيرات (لأننا لن نسير إليها لاحقا) : لذا يجب علينا تطبيق الأسلوب **pack()** مباشرة في اللحظة التي نصنع فيها هذه الكائنات.

الشيء الجديد في هذا البرنامج هو الدالة **avance()** التي تم تعريفها في بداية السكريبت. في كل مرة يتم إستدعائها، تقوم هذه الدالة بإعادة تحديد إحداثيات كائن "الدائرة الملونة" التي تم وضعها في للوحة، مما يتسبب في تحريك هذا الكائن.

هذه الطريقة تتميز بها البرمجة الشيءية، التي تبدأ من صنع الكائنات ثم يتم العمل على هذه الكائنات من خلال تغيير خصائصها، من خلال الأساليب.

في البرمجة الحتمية "القديمة" (و هذا يعني بدون استخدام الكائنات). يتم تحريك الأرقام عن طريق حذفها ثم إعادة رسمها في مكان أبعد قليلا. أما في البرمجة الشيئية، يتم التعامل مع هذه المهام تلقائيا من قبل الفئات التي يتم اشتقاق الكائنات منها، حتى لا يتم تضييع الوقت في إعادة برمجة.

تمارين

12.8 اكتب برنامجا يظهر نافذة بها لوحة. في هذه اللوحة يجب أن تكون بها دائرتان (بلونين وحجمين مختلفين)، والتي من المفترض أن يمثل كوكبين. وأزرار تسمح لنقلهم إلى volonté كليهما إلى جميع الاتجاهات. تحت اللوحة، يجب على البرنامج أن يظهر دائما أ) المسافة بين الكوكبين، ب) قوة الجاذبية التي يقوم بها كل واحد ضد الأخر (تستطيع أن تظهر في أعلى النافذة الكتل المختارة لكل واحد منهما، ومسافة النطاق). في هذا التمرين، يجب عليكم استخدام قانون نيوتن للجاذبية (التمرين 6.16، الصفحة 61، والدليل الفيزيائي العام.

13.8 استوح من برنامج يكتشف نقرات الفأرة في اللوحة، عدل البرنامج المذكور أعلاه وذلك للحد من عدد الأزرار : لوضع الكوكب، يتم ببساطة باختيار زر، ثم يتم الضغط على اللوحة ليتم وضع الكوكب في المكان الذي يتم النقرة عليه.

14.8 امتدادا للبرنامج أعلاه. ضع كوكب آخر، وأعرض القوة المؤثرة على الثلاثة (في الحقيقة: كل واحد وفي جميع الأوقات قوة الجاذبية التي يبذلها من اثنين أخرين) .

15.8 نفس التمرين مع الشحنات الكهربائية (قانون كولوم). أعطِ هذه المرة لاختيار الشحنات.

16.8 اكتب برنامجا صغيرا يظهر نافذة مع حقلين : الأول يشير إلى درجة الحراة المئوية، والأخر درجة فهرنهايت. في كل مرة يتم تغيير أي واحدة من الدرجات الحرارة، يتم تصحيح الأخرى وفقا لذلك. لتحويل الفهرنهايت إلى درجة مئوية،

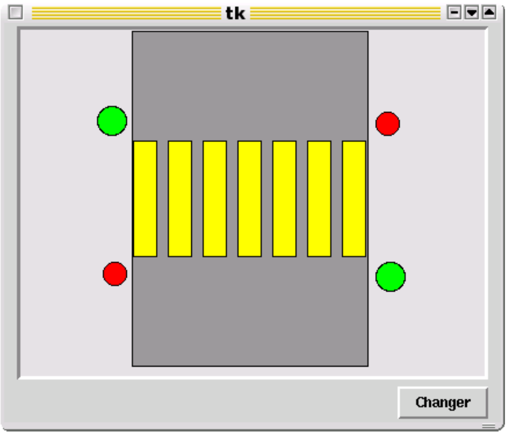
والعكس بالعكس، يتم بذلك باستخدام المعادلة $T_F = T_C \times 1,80 + 32$. يمكنك أيضا مراجعة برنامج الآلة الحاسبة البسيطة (الصفحة 96).

17.8 اكتب برنامجا يظهر نافذة بها لوحة. في هذه اللوحة، ضع دائر صغيرة التي من المفترض أن تكون كرة. تحت اللوحة، ضع زر. في كل مرة يتم الضغط على الزر، تتقدم الكرة مسافة قصير إلى اليمين حتى تصل إلى نهاية اللوحة. فإذا لازلت تضغط على الزر، سوف ترجع الكرة إلى الطرف الأخر، وهكذا.

18.8 حسن البرنامج أعلاه لكي تتحرك الكرة بشكل دائري أو بيضوي في اللوحة (عند النقر مرارا وتكرارا). ملاحظة : للحصول على النتيجة المطلوبة، سوف تقوم بالضرورة بتعريف متغير لتمثيل الزاوية، واستخدام الدالتين sinus و cosinus لوضع الكرة لوفقا لهذه الزاوية.

19.8 عدل البرنامج الذي بالأعلى بطريقة تجعل الكرة، عندما تتحرك تترك وراءها أثرا من المسار.

20.8 عدل البرنامج المذكور أعلاه بطريقة لتوجيه أرقام أخرى .استشر أستاذك للحصول على اقتراحات (أرقام Lissajous).



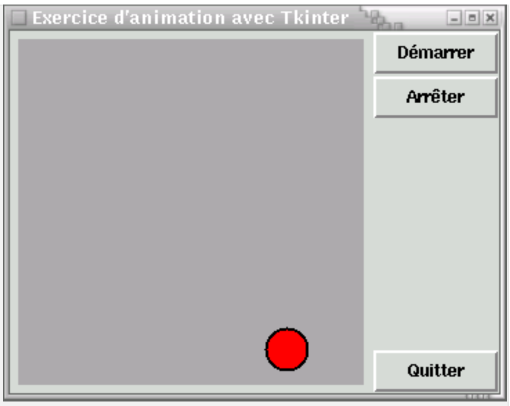
21.8 اكتب برنامجا يظهر نافذة مع لوحة وزر. في اللوحة، ارسم مستطيلا رماديا غامقا، والذي يمثل الطريق، ثم قم برسم سلسلة مستطيلات صفراء تمثل ممرا لعبور المشاة. أضف أربعة دوائر ملونة تشير إلى إشارات المرور للمشاة والسيارات. ومع كل ضغطة على الزر سوف يتغير لون الأضواء.

اكتب البرنامج الذي يظهر لوحة مرسوم عليها دارة كهربائية بسيطة (مولد + مبدل + مقاوم). ويجب أن يكون في النافذة حقول إدخال لإدخال برامتر كل عنصر. (و هذا معناه تحديد قيم المقاومة والفولتية). ويجب أن يكون المبدل يعمل (بزر اعمل\توقف). بالإضافة للمصقات (لابل - Label) يجب أن تعرض دائما الفولتية والتيارات الناجمة عن الخيارات التي قام بها المستخدم.

رسوم متحركة تلقائية

وفي الختام هذه أول مرة نتصل مع واجهة الرسومية tkinter، هذا هو آخر مثال لرسوم متحركة، والذي يعمل بشكل مستقل عند الضغط على "اعمل" .

```
from tkinter import *
```




```
# تعريف متلقي الأحداث :

def move():
    "déplacement de la balle"
    global x1, y1, dx, dy, flag
    x1, y1 = x1 +dx, y1 + dy
    if x1 >210:
        x1, dx, dy = 210, 0, 15
    if y1 >210:
        y1, dx, dy = 210, -15, 0
    if x1 <10:
        x1, dx, dy = 10, 0, -15
    if y1 <10:
        y1, dx, dy = 10, 15, 0
    can1.coords(oval1,x1,y1,x1+30,y1+30)
    if flag >0:
        fen1.after(50,move)          # => ضعه في الحلقة بعد 50 ميلي ثانية

def stop_it():
    "arrêt de l'animation"
    global flag
    flag =0

def start_it():
    "démarrage de l'animation"
    global flag
    if flag ==0:             # لكي لا يتم تشغيل سوى حلقة واحدة
       flag =1
       move()

#==========  البرنامج الرئيسي =============

# سيتم استخدام هذه المتغيرات كمتغيرات عامة :
x1, y1 = 10, 10              # الإحداثيات الأولية
dx, dy = 15, 0               # خطوة الإزاحة
flag =0                      # العدّاد

# صنع ودجة الرئيسية الأصل :
fen1 = Tk()
fen1.title("Exercice d'animation avec tkinter")
# صنع ودجة الأطفال :
can1 = Canvas(fen1,bg='dark grey',height=250, width=250)
can1.pack(side=LEFT, padx =5, pady =5)
oval1 = can1.create_oval(x1, y1, x1+30, y1+30, width=2, fill='red')
bou1 = Button(fen1,text='Quitter', width =8, command=fen1.quit)
bou1.pack(side=BOTTOM)
bou2 = Button(fen1, text='Démarrer', width =8, command=start_it)
bou2.pack()
bou3 = Button(fen1, text='Arrêter', width =8, command=stop_it)
bou3.pack()
# بدء متلقي الأحداث (الحلقة الأساسية) :
fen1.mainloop()
```

الشيء الجديد في هذا السكريبت يقع عند نهاية تعريف الدالة **move()** : هل لاحظت استخدام الأسلوب **after()**. يمكن تطبيق هذا الأسلوب على أي ودجة. هذا يتسبب في استدعاء الدالة بعد فترة زمنية معينة. على سبيل المثال، (`window.after(200,qqc`) تقوم الودجة **window** باستدعاء الدالة **qqc()** بعد توقف دام 200 ميلي ثانية.

في السكريبت الخاص بنا، الدالة التي تم استدعاؤها من قبل الأسلوب **after()** هي الدالة **move()** نحن استخدمنا هنا للمرة الأولى تقنية برمجة قوية جدا وتدعى (الاستدعاء الذاتي - récursivité). لجعله بسيط، نقول إن الاستدعاء الذاتي هو ما يحدث عندما تستدعي الدالة نفسها. وبالطبع سوف نحصل على حلقة يمكن أن تستمر إلى ما لانهاية إذا لم تضع طريقة لوقفها.

دعونا نرى كيف يعمل في مثالنا.

يتم استدعاء الدالة **move()** للمرة الأولى عندما يتم النقر على زر البدء (Démarrer). ستبدأ عملها (و هذا معناه وضع الكرة). ثم، من خلال الأسلوب **after()**ـ التي يتم استدعاؤها بعد فاصل قصير. ثم تبدأ الدورة الثانية، التي سوف تستدعي نفسها مرة أخرى، وهكذا إلى أجل غير مسمى.

هذا على الأقل ما سيحدث إذا لم نتخذ الاحتياطات اللازمة لوضع تعليمة الإخراج في مكان ما. في هذه الحالة، هذا اختبار شرطي بسيط : في كل حلقة، نحن نفحص محتويات المتغير **flag** بمساعدة العبارة **if**. فإذا كان محتوى المتغير **flag** سوف تتوقف الحلقة ويتوقف تحريك الرسوم. لاحظ أننا حرصنا على تعريف المتغيرـ **flag** على أنه متغيرـ عام. وبالتالي يمكننا تغيير قيمته بسهولة بمساعدة دالات أخرى، مثل تلك المرتبطة بأزرار بدء وإيقاف.

حصلنا على ألية بسيطة لتشغيل وإيقاف الرسوم المتحركة : عند الضغطة الأولى على زر البدء سيتم تعيين قيمة ليست لصفر للمتغير **flag**ـ ثم سيتم استدعاء لأول مرة الدالة **move()**. التي ستعمل، وستستدعي نفسها كل 50 ميلي ثانية، إلى أن يكون قيمة المتغير **flag** صفر. فإذا استمررت بالضغط على زر البدء ، لن يتم استدعاء الدالة **move()** لأن قيمة المتغيرـ **flag** هي 1. ولذلك سوف تبدأ حلقة مرات عديدة.

الزر توقف (Arrêter) يعين للمتغير **flag** القيمة صفر، وتتوقف الحلقة .

تمارين

22.8 في الدالة **start_it()**، احذف التعليمة `if flag == 0:` (و السطرين التاليين اللذين يبدأن بمسافة البادئة). ماذا حدث ؟ (اضغط مرات عديدة على زر البدء. حاول أن تتكلم بأكبر قدر من الوضوح لتفسير لماذا حدث .

23.8 عدل البرنامج بحيث يتغير لون الكرة في كل دورة.

24.8 عدل البرنامج بحيث تجعل حركات الكرة منحرفة كما كرة البلياردو (متعرج).

25.8 عدل البرنامج لتحصل على حركات أخرى. على سبيل المثال حركة دائرية (كما في التمرين صفحة 109).

26.8 عدل البرنامج، أو اكتب واحدا مشابها له، لمحاكاة سقوط الكرة (بسبب الجاذبية) وارتدادها. تنبيه : هذه المرة يجب أن تكون الحركة متسارعة (تزداد السرعة مع الوقت).

27.8 بداية من السكريبتات المذكورة أعلاه، يمكنك الآن كتابة لعبة تعمل على النحو التالي : كرة تتحرك بشكل عشوائي على اللوحة، بسرعة بطيئة. يجب على اللاعب الضغط على الكرة باستخدام الفأرة. فإذا نجح، يحصل على نقطة، لكن الكرة تزداد سرعتها، وهكذا. أوقف اللعبة بعد عدد معين من النقرات وقم بعرض النتيجة .

28.8 غير من السكريبت السابق : في كل مرة ينجح فيها اللاعب في القبض على الكرة يصبح حجمها أصغر (يمكن أيضا تغيير لونها).

29.8 اكتب البرنامج الذي به العديد من المرات مختلفة الألون، والتي تقفز في كل مكان، وعلى الجدران .

30.8 اصنع لعبة مثالية من خلال السكريبتات السابقة، من خلال دمج الخوارزميات أعلاه. يجب على اللاعب الضغط فقط على الكرات الحمراء. وإذا نقر بالخطأ على كرة من لون آخر يفقد بضعة نقاط.

31.8 اكتب البرنامج الذي يحاكي اثنين من الكواكب التي تدور حول الشمس في مدارات دائرية مختلفة (أو إثنين من الألكترونات التي تدور حول نواة الذرة ...).

32.8 اكتب برنامجا للعبة الثعبان : ثعبان (يتكون من خط قصير من المربعات) يتحرك على اللوحة في أربعة اتجاهات : يسار ويمين وأعلى وأسفل. يمكن للاعب تغيير اتجاه الثعبان من خلال الأسهم في لوحة المفاتيح. وفي القماش يوجد أيضا "الفريسة" (دوائر صغيرة مرتبة عشوائيا). يجب على الثعبان أن يأكل الفريسة دون أن يصطدم مع حواف اللوحة. في كل مرة يأكل فيها فريسة يزداد الثعبان طولا، ويربح اللاعب نقطة ، وتظهر الفريسة جديدة في مكان آخر. اللعبة تتوقف عندما يلمس الثعبان أحد الجدران أو عندما يصل إلى طول محدد.

33.8 طور اللعبة السابقة بإضافة : تتوقف اللعبة إذا تداخل الثعبان .

9

التعامل مع الملفات

حتى الآن، جميع البرامج التي صنعناها لا تتعامل سوى مع كمية صغيرة جدا من البيانات. وفي كل مرة نريد أن نتعامل مع هذه البيانات نضعها في جسم البرنامج نفسه(على سبيل المثال في قائمة). هذه الطريقة غير كافية إلى حد بعيد عندما نريد التعامل مع كمية أكبر ومهمة من المعلومات .

فائدة الملفات

لنفترض على سبيل المثال أننا نريد كتابة برنامج صغير يظهر على الشاشة أسئلة متعددة الخيارات، مع معالجة تلقائية لردود المستخدم. كيف يمكننا تخزين نص الأسئلة ؟

أبسط فكرة هي وضع كل نص في متغير، في بداية البرنامج، مع عبارات التعيين، مثلا :

```
a = "Quelle est la capitale du Guatemala ?"
b = "Qui à succédé à Henri IV ?"
c = "Combien font 26 x 43 ?"
        ... etc.
```

للأسف هذه الفكرة بسيطة جدا. وسيصبح باقي البرنامج معقدا جدا، هذا معناه أن التعليمات التي سيتم استخدامها لاختيار سؤال أو أكثر بشكل عشوائي ليتم تقديمه للمستخدم. سوف تستخدم على سبيل المثال مجموعة طويلة من عبارات **if** ... **elif** ... **elif** ... كما في المثال في الأسفل وهو بالتأكيد ليس الحل الجيد (و سيكون متعبا جدا في الكتابة، لا تنسى أيضا كتابة معالجة (إجابة) كل هذه الأسئلة!) :

```
if choix == 1:
    selection = a
elif choix == 2:
    selection = b
elif choix == 3:
    selection = c
        ... etc.
```

سيكون أفضل إذا استخدمنا القوائم :

```
liste = ["Qui a vaincu Napoléon à Waterloo ?",
         "Comment traduit-on ’informatique’ en anglais ?",
         "Quelle est la formule chimique du méthane ?", ... etc ...]
```

يمكن للمرء أن يستخرج أي عنصر من هذه القائمة باستخدام المؤشر. على سبيل المثال :

```
print(liste[2])        ===>  "Quelle est la formule chimique du méthane ?"
```

> تذكير
>
> العداد يبدأ من الصفر

في حين أن هذه الطريقة أفضل بكثير من الطريقة السابقة، لكن مازلنا نواجه العديد من المشاكل المزعجة :

- إن قابلية قراءة هذا البرنامج تتدهور بسرعة كبيرة عندما يصبح عدد الأسئلة كبير جدا. وطبعا، سوف نزيد من احتمالية إدراج خطأ في تعريف هذه القائمة الطويلة . بعض الأخطاء قد يصعب جدا العثور عليها.
- إضافة أسئلة جديدة، أو تعديل على الأسئلة الموجودة ، يجب علينا في كل مرة فتح النص المصدري للبرنامج. وطبعا سيصعب إعادة صياغة التعليمات البرمجية في النص المصدري، لأنه يشمل العديد من الأسطر التي بها معطيات معقدة.
- تبادل البيانات مع برامج أخرى (ربما كتبت بلغات برمجة أخرى) هو بكل بساطة مستحيل، لأن هذه البيانات هي جزء من البرنامج نفسه .

ملاحظة أخيرة : حان الوقت لنتعلم فصل البيانات والبرامج التي تعالجها في ملفات مختلفة.

ليكون هذا ممكنا، سوف نقدم مجموعة متنوعة من الآليات لإنشاء الملفات وإرسال البيانات وإسترجاعها في وقت آخر.

لغات البرمجة تعرض تعليمات أكثر أو أقل تعقيدا لأداء هذه المهام. عندما يتم التعامل مع مجموعات كبيرة من البيانات، يكون من المهم (ضروري) تنظيم العلاقة بين هذه البيانات ، وعلينا وضع أنظمة تسمى بقواعد البيانات، يمكن أن تكون إدارتها معقدة للغاية. عندما تواجه مشاكل من هذا القبيل، يجب أن تفوض هذا إلى العديد من البرامج المتخصصة في هذا مثل : Oracle، MySQL، PostgreSQL، Adabas، Sybase، IBM DB2 ....إلخ. بيثون يمكنها التواصل مع هذه الأنظمة، لكن سنترك هذا لوقت آخر (انظر : إدارة قواعد البيانات، صفحة 302).

طموحاتنا متواضعة جدا. البيانات التي لدينا ليست بمئات الآلاف، نكتفي بألية بسيطة لحفظ البيانات في ملف متوسط الحجم، ومن ثم استخراجها عندما نحتاجها .

العمل مع الملفات

استخدام الملف يشبه إلى حد كبير استخدام كتاب. لاستخدام كتاب، يجب أن تجده أولا (بمساعدة اسمه)، ثم يجب عليك فتحه. وعندما تنتهي من استخدامه، تقوم بإغلاقه. وعندما يكون مفتوحا، يمكنك قراءة المعلومات المختلفة، ويمكنك أيضا كتابة تعليقات توضيحية، لكن عموما لن تقوم بعمل الاثنين في وقت واحد. في جميع الحالات، يمكنك معرفة أين وصلت في داخل الكتاب، وذلك بمساعدة أرقام الصفحات. تقرأ معظم الكتاب باتباع نظام الصفحات، لكن يمكنك أيضا قراءة أي فقرة بشكل مضطرب.

كل ما قلناه عن الكتب ينطبق أيضا على ملفات الحاسوب .الملف يتكون من بيانات مخزنة على القرص الصلب، أو في قرص مرن، أو في إصبع usb أو في قرص مدمج (cd). والتي يمكنك الوصول إليها من اسمها (و قد يشمل اسم الدليل). كنظرة تقريبية، قد تنظر إلى محتويات الملف كأنه سلسلة من الأحرف، مما يعني أنه يمكنك التعامل مع هذا المحتوى، أو أي جزء منه، بمساعدة دالات تتعامل مع سلاسل الأحرف[41].

اسم الملف - الدليل الحالي

لتبسيط التفسيرات التي تلي، سوف نقوم بتوضيح فقط الأسماء البسيطة للملفات التي سوف تعامل معها. إذا كنت تفعل هذا في تمارينك، الملفات سوف يتم صنعها و\أو يتم البحث عنها من قبل بيثون في الدليل الحالي. عادة ما يكون هذا الدليل في نفس مكان السكريبت، إلا إذا كنت تشغل السكريبت من خلال نافذة ال IDLE، في هذه الحالة، يتم تعيين الدليل الحالي عند تشغيل ال IDLE (في ويندوز تعريف هذا الدليل هو جزء من خصائص أيقونة التشغيل).

إذا كنت تعمل مع IDLE، فإنك بالتأكيد تريد إجبار بيثون تغيير الدليل الحالي، بحيث يكون مثلما تريده. وللقيام بذلك، يجب عليك كتابة الأوامر التالية عند بداية الحصة. نحن نفترض أن الدليل الذي تريده هو home/jules/exercices/. حتى لو كنت تعمل في نظام ويندوز (وهذه ليست قاعدة)، يمكنك استخدام نفس التعليمة (ولكن يجب عليك استخدام \ بدل من / : لأن الأولى تعمل فقط على أنظمة UNIX). بيثون سيقوم بالتحويلات اللازمة، هذا إذا كنت تعمل على Mac OS أو Linux أو Windows[42].

```
>>> from os import chdir
>>> chdir("/home/jules/exercices")
```

[41] بعبارة أدق يجب عليك أن تأخذ بعين الاعتبار أن محتوى الملف هو تسلسل من البايتات. معظم البايتات ممثلة بواسطة رموز والعكس غير صحيح : سوف تتعلم في نهاية المطاف التمييز الواضح بين سلاسل البايتات والسلاسل النصية .

[42] في حالة استخدامك لنظام تشغيل ويندوز يمكنك تضمين في هذا المسار الرسالة التي تعين جهاز التخزين حيث يوجد الملف. على سبيل المثال: D:/home/jules/exercices.

الأمر الأول يستدعي الدالة **chdir()** من وحدة **os**. تحتوي وحدة **os** على جميع الدالات التي تتعامل مع أنظمة التشغيل (**os** = *operating system*)، أي نظام تشغيل بغض النظر عن نوعه.

الأمر الثاني يتسبب في تغيير الدليل (**chdir** =*change directory*).

> • يمكنك أيضا إدراج هذه الأوامر (التعليمات) في بداية البرنامج النصي، أو تحديد اسم المسار الكامل للملف الذي تريد معالجته، ولكن هذا قد يكون خطر على ثقل الكتابة في برامجك.
>
> • اختر أسماء ملفات قصيرة. تجنب قدر الإمكان الأحرف المعلمة والمسافات والعلامات المطبعية الخاصة. في بيئات عمل اليونكس (ماك، لينكس، *BSD* ...)، ينصح في أغلب الأحيان باستخدام الأحرف الصغيرة فقط .

شكلي الاستدعاء.

أسطر التعليمات التي سوف نستخدمها هي فرصة لشرح آليات مثيرة للاهتمام. أنت تعرف أنه بالإضافة إلى الدالات المدمجة في الوحدات الأساسية، بيثون يوفر لك كمية هائلة من الدالات الخاصة، والتي تم تجميعها في وحدات. من الوحدات التي عرفتها الوحدة math والوحدة tkinter.

لاستخدام الدالات من وحدة، يجب عليك استدعائها. لكن هذا يتم بطريقتين مختلفتين، كما سنرى بالأسفل، كل واحدة لديها مميزاتها وعيوبها.

هذا مثال على الطريقة الأولى :

```
>>> import os
>>> rep_cour = os.getcwd()
>>> print rep_cour
C:\Python22\essais
```

في السطر الأول من هذا المثال نحن نستدعي الوحدة **os**، التي تحتوي على وظائف كثيرة مثيرة للاهتمام التي تسمح لنا بالوصول إلى نظام التشغيل. أما السطر الثاني فهو يستخدم الدالة **getcwd()** من وحدة **os**[43]. كما ترون، الدالة **getcwd()** تقوم بإرجاع اسم الدليل الحالي (*getcwd = get current working directory*). للمقارنة، هنا مثال على الطريقة الثانية :

```
>>> from os import getcwd
>>> rep_cour = getcwd()
```

---

43 النقطة الفاصلة تعبر هنا عن علاقة الانتماء. وهذا مثال على الأسماء المؤهلة التي سيتم استخدامها على نطاق واسع في ما تبقى من هذه الدورة. ربط الأسماء بمساعدة النقاط هي وسيلة لا بأس بها للعناصر التي تشكل جزءا من المجموعات، والتي هي ربما قد تكون من مجموعة أكبر وإلخ. على سبيل المثال، تسمية `systeme.machin.truc` تشير إلى عنصر `truc`، والذي هو جزء من المجموعة `machin`، والذي هو جزء من المجموعة `systeme`. سوف نرى العديد من الأمثلة عن هذه التقنية، وخصوصا عندما ندرس أصناف الكائنات .

```
>>> print(rep_cour)
C:\Python31\essais
```

في هذا المثال الجديد، قمنا باستدعاء الدالة **getcwd** فقط من الوحدة **os**. بالاستدعاء بهذه الطريقة، سيتم دمج الدالة مع كود كما لو أننا كتبناه بأنفسنا. في الأسطر التي نستعملها، ليس من الضروري ذكر جزء الوحدة **os**.

و يمكننا استدعاء العديد من الدالات من نفس الوحدة بنفس الطريقة :

```
>>> from math import sqrt, pi, sin, cos
>>> print(pi)
3.14159265359
>>> print(sqrt(5))          # الجذر التربيعي لـ 5
2.2360679775
>>> print(sin(pi/6))        # جيب الزاوية 30 درجة
0.5
```

و يمكننا استدعاء كل الدالات من وحدة ، كما في :

```
from tkinter import *
```

لهذه الطريقة ميزة سهولة كتابة التعليمات البرمجية للدالات التي تم إستدعائها. ولديها أيضا عيب (خاصة في الشكل الأخير عند استدعاء جميع دالات الوحدة) تشوش مساحة الاسم الحالي. قد تكون بعض الدالات التي استدعيتها لديها نفس اسم المتغير الذي عرفته أنت، أو نفس اسم دالة تم إستدعائها من وحدة أخرى. فإذا حدث هذا، واحد من الاسمين المتضاربين لن يتم الوصول إليه بشكل جيد.

في البرامج التي لديها بعض الأهمية، والتي تستدعي عددا كبيرا من وحدات من مختلف المصادر، سيكون من الأفضل لها أن تستخدم الطريقة الأولى، وهذا معناه استخدام أسماء مؤهلة بشكل كامل.

عموما، يوجد استثناءات لهذه القاعدة في حالة معينة من الوحدة tkinter، لأنه يحتوي على دالات مطلوبة بشدة (عندما تقرر استخدام هذه الوحدة).

كتابة متسلسلة في ملف

في بيثون، يتم توفير الوصول إلى الملفات عن طريق كائن الواجهة خاصة، التي تسمى كائن الملف. نحن نقوم بصنع هذا الملف باستخدام الدالة المدمجة **open()**[44]. التي تقوم بإرجاع الكائن مع أساليب محددة، والتي تسمح لك بقراءة وكتابة في هذا الملف.

44هذه الدالة. هي قيمة رجوع لكائن معين. وغالبا ما تسمى مصنع الدالة

المثال التي يوضح كيفية فتح ملف ، ثم كتابة سلسلتين نصيتين فيه، ثم إغلاقه. لاحظ أن إذا كان الملف غير موجود فسوف ستم إنشاءه تلقائيا. ومن جهة أخرى، وإذا كان الملف موجود بالفعل وبه بعض البيانات، الحروف التي سوف تسجلها ستكون بعد الموجودة. يمكنك أن تقوم بهذا التمرين مباشرة على سطر الأوامر :

```
>>> obFichier = open('Monfichier','a')
>>> obFichier.write('Bonjour, fichier !')
>>> obFichier.write("Quel beau temps, aujourd'hui !")
>>> obFichier.close()
>>>
```

ملاحظات

- السطر الأول يقوم بإنشاء ملف الكائن **obFichier**، والتي ستشير (مرجع) إلى الملف الحقيقي(على القرص أو على القرص المرن) وهو سيكون اسمه **Monfichier**. تنبيه : لا تخلط بين اسم الملف مع اسم كائن الملف الذي يتيح الوصول إليه ! بعد هذا التمرين، يمكنك التحقق من أن هذا الملف تم إنشاؤه في نظامك (في الدليل الحالي) اسم الملف هو **Monfichier** (و الذي تستطيع عرض محتواه مع أي محرر) .
- الدالة **open()** تنتظر برامترين[45]، والذين يجب أن يكونا سلسلتين نصيتين. البرامتر الأول سيكون اسم الملف الذي يجب فتحه، والثاني هو اسم وضع الفتح. 'a' يشير إلى فتح الملف بوضع إضافة (append)، وهذا يعني أن البيانات التي سيتم حفظها سيتم إضافتها إلى نهاية الملف، إضافة إلى التي هي موجودة بالفعل. نستطيع أيضا استخدام الوضع 'w' (للكتابة - write)، لكن عند استخدام هذا الوضع، سيقوم بيثون بصنع ملف جديد (فارغ)، ويكتب فيه البيانات، بداية من بداية الملف. فإذا وجد ملفًا بنفس الاسم، يتم مسحه وصنع ملف جديد.
- الأسلوب **write()** يكتب فعليا. ويجب أن تكون البيانات التي يجب كتابتها كبرامتر. هذه البيانات يتم حفظها في الملف واحدًا بعد الآخر (نحن نتحدث هنا عن الوصول المتسلسل للملف). عند كل استدعاء للدالة **write()** تستمر الكتابة في الملف (مع الموجود بالفعل).
- الأسلوب **close()** تغلق الملف. وهي متاحة لجميع الاستعمالات .

قراءة متسلسلة من الملف

سوف نقوم الآن بإعادة فتح الملف، لكن هذه المرة، من أجل قراءة المعلومات التي سجلناها في الخطوة السابقة :

```
>>> ofi = open('Monfichier', 'r')
>>> t = ofi.read()
>>> print(t)
Bonjour, fichier !Quel beau temps, aujourd'hui !
>>> ofi.close()
```

[45] يمكن إضافة برامتر ثالث للإشارة إلى الترميز المستخدم (انظر إلى صفحة 143).

كما كان متوقعا، الأسلوب **read()** يقرأ البيانات في الملف ويحوله إلى متغير من نوع سلسلة نصية(string). فإذا استخدمنا هذا الأسلوب بدون برامترات، يتم نقل كامل محتويات الملف .

ملاحظات

- الملف الذي نريد استدعائه لقرائته يدعى **Monfichier**. يجب علينا أن نكتب تعليمة فتح الملف ليشير إلى الملف. فإذا كان الملف غير موجود سوف تحصل على رسالة خطأ. على سبيل المثال :

```
>>> ofi = open('Monficier','r')
IOError: [Errno 2] No such file or directory: 'Monficier'
```

- من جهة أخرى، نحن غير ملزمين باختيار اسم محدد لكائن الملف. نستطيع اختيار اسم أي متغير. وبالتالي في تعليمتنا الأولى نحن قمنا بصنع كائن ملف وسميناه **ofi**، والذي سيكون إشارة للملف الأصلي **Monfichier**، والذي سوف يتم فتحه للقراءة منه (البرامتر 'r').
- السلسلتان النصيتان اللتان وضعناهما في الملف تم دمجهما في سطر واحد. هذا طبيعي، لأننا لم نستخدم أي رمز خاص عندما قمنا بحفظه. سوف نتعرف لاحقا على طريقة حفظ أسطر منفصلة
- الأسلوب **read()** يمكننا استخدامه مع برامتر. وسوف يشير إلى عدد الحروف التي يجب أن تقرأ. بداية من الموقع الموجود بالفعل في الملف :

```
>>> ofi = open('Monfichier', 'r')
>>> t = ofi.read(7)
>>> print(t)
Bonjour
>>> t = ofi.read(15)
>>> print(t)
, fichier !Quel
```

فإذا لم يكن ما يكفي من الحروف في الملف، تتوقف القراءة في نهاية الملف :

- ```
  >>> t = ofi.read(1000)
  >>> print(t)
   beau temps, aujourd'hui !
  ```

فإذا تم الوصول بالفعل إلى نهاية الملف، فإن **read()** تقوم بإرجاع سلسلة فارغة :

```
>>> t = ofi.read()
>>> print(t)
```

- لا تنسَ إغلاق الملف بعد استعماله

```
>>> ofi.close()
```
```

> في كل ما سبق، لقد افترضنا دون شرح أن السلاسل النصية يتم تبادلها بين مفسر بيثون والملف. وهذا في الواقع غير صحيح، لأن السلاسل النصية يتم تحويلها إلى سلاسل بايتات ليتم تخزينها في ملفات. وبالإضافة إلى ذلك، هنالك للأسف معايير مختلفة لهذا الغرض. بالمعنى الدقيق، في بيثون ينبغي أن يكون واضحا معيار الترميز الذي يجب استخدامه في ملفاتك : سنرى كيف يمكننا فعل هذا في الفصل القادم. في غضون ذلك، يمكنك الاعتماد على بيثون لأن بيثون يستخدم المعيار الافتراضي لنظامك. وهذا سوف يجنبك المشاكل في التمارين الأولى. ومع ذلك، إذا كانت بعض المعلمات تظهر بشكل غريب، يرجى تجاهل هذا مؤقتا .

العبارة break للخروج من الحلقة

ليس علينا القول أننا بحاجة إلى حلقات في البرنامج عندما نتعامل مع ملف لا نعرف محتوياته. والفكرة هي قراءة الملف جزءاً جزءاً حتى نصل إلى نهاية الملف.

الدالة التي بالأسفل تشرح هذه الفكرة. فهي تقوم بنسخ ملف بأكمله (بغض النظر عن حجمه) من خلال نقل 50 حرف في كل مرة :

```
def copieFichier(source, destination):
    "copie intégrale d'un fichier"
    fs = open(source, 'r')
    fd = open(destination, 'w')
    while 1:
        txt = fs.read(50)
        if txt =="":
            break
        fd.write(txt)
    fs.close()
    fd.close()
    return
```

فإذا أردت اختبار هذه الدالة، يجب عليك توفير دالتين : الأول اسم الملف الأصلي، والثانية اسم الملف الذي تريد الاستنساخ إليه. على سبيل المثال :

```
copieFichier('Monfichier','Tonfichier')
```

ستلاحظ أننا عندما قمنا ببناء الحلقة، استخدمنا طريقة مختلفة عن التي عرفناها سابقا. كما تعلم أن العبارة **while** يجب دائما أن تلحقها الشرط لتقييمه ، ويتم تنفيذ الكتلة التي تليه في الحلقة، مادامت هذه الدالة صحيحة. ولكننا هنا قمنا باستبدال الشرط التحقق برقم بسيط، وأنت تعرف[46] أن مفسر بيثون يعتبر أي قيمة غير الصفر قيمة صحيحة.

حلقة **while** بالأعلى سوف تعمل لأجل غير مسمى، لأن الشرط يبقى دائما صحيحا. ومع ذلك، يمكننا أن نقطع هذه الحلقة باستخدام عبارة **break**، وربما يجب علينا أن نضع عدة عبارات توقف في عدة أماكن :

46انظر إلى صفحة 56 : **تعبير حقيقي\مزيف**

```
while <condition 1> :
    --- instructions diverses ---
    if <condition 2> :
        break
    --- instructions diverses ---
    if <condition 3>:
        break
    etc.
```

في دالتنا **copieFichier()**، من السهل أن يتم تنفيذ العبارة **break** بعد إنهاء قراءة الملف .

ملفات نصيّة

الملف النصي هو ملف يحتوي على أحرف "للطباعة - imprimables"[47] وفراغات تنظم الأسطر، هذه الأسطر يتم فصلها عن بعض عن طريق رمز خاص غير مطبوع يسمى علامة نهاية السطر.[48]

الملفات النصية هي الملفات التي نستطيع قراءتها وفهمها مع محرر نص بسيط، بدلا من الملفات الثنائية - على الأقل في جزء منه - غير المفهومة للبشر. ويكون له معنى فقط عندما يتم فك شيفرته من قبل برامج خاصة. على سبيل المثال، الملفات التي تحتوي على صور وصوتيات وفيديوات ...إلخ. هم دائما تقريبا ملفات ثنائية. أعطينا مثالا صغيرا على التعامل مع الملفات الثنائية ، لكن في هذه الدورة، سوف نركز فقط على الملفات النصية.

من السهل جدا معالجة الملفات النصية مع بيثون. على سبيل المثال، هذه التعليمات كافية لصنع ملف نصي يتكون من أربعة أسطر :

```
>>> f = open("Fichiertexte", "w")
>>> f.write("Ceci est la ligne un\nVoici la ligne deux\n")
>>> f.write("Voici la ligne trois\nVoici la ligne quatre\n")
>>> f.close()
```

لاحظ أن في نهاية السطر **\n** حيث يتم إدراجها في السلاسل النصية، حتى نفصل بين أسطر النص السابق عندما نقوم بحفظه. من دون هذه العلامة، سوف يتم حفظ الحروف واحدا بعد الآخر، كما في الأمثلة السابقة.

عند قراءة الملف، أسطر الملف يمكن أن يتم استراجاعها بشكل منفصل عن بعضهم. عن طريق الأسلوب **readline()**. على سبيل المثال، لن يقرأ هنا سوى سطر واحد في كل مرة (بما في ذلك علامة نهاية السطر) .

47 بالمعنى الدقيق، وكما تناولنا ملف يحتوي على "بايتات قابلة للطباعة" التي قيمها يمكنها أن تمثل رموز طباعية في ترميز محدد جدا. سوف نناقش هذا بتفصيل أكثر في الفصل القادم. تحديدا، هو بيتات ذات قيمة رقمية ما بين 32 و 255. والبايتات للقيمة أقل من 32 هي رموز قديمة للتحكم وعموما لا يمكن تمثيلها بواسطة أحرف .

48 اعتمادا على نظام التشغيل المستخدم، الترميز الموافق لنهاية السطر قد يكون مختلفا. وعلى سبيل المثال، في نظام تشغيل ويندوز هنالك تسلسل إثنين من البايتات (رمز الإرجاع وقفز السطر)، في حين أن أنظمة التشغيل من نوع يونكس (مثل لينكس) يكفي أن تضع القفز السطر. أما في "ماك" فهو يستخدم رمز الرجوع. من حيث المبدأ، لا يوجد ما يدعو للقلق حول هذه الاختلافات. من خلال عمليات الكتابة، يستخدم بيثون اتفاقية موجودة في نظامك. للقراءة، يقوم بيثون بتصحيحها حسب الإتفاقية (ما يعادلها) .

```
>>> f = open('Fichiertexte','r')
>>> t = f.readline()
>>> print(t)
Ceci est la ligne un
>>> print(f.readline())
Voici la ligne deux
```

الأسلوب **readlines()** ينقل جميع الأسطر المتبقية في سلسلة نصية : .

```
>>> t = f.readlines()
>>> print(t)
['Voici la ligne trois\n', 'Voici la ligne quatre\n']
>>> f.close()
```

ملاحظات

- في القائمة أعلاه تظهر في شكلها الخام، مع علامة اقتباس واحدة للسلاسل، وحروف خاصة في شكلها التقليدي. ويمكنك بالطبع استعراض هذه القائمة (بمساعدة الحلقة **while**، على سبيل المثال) لاستخراج السلاسل الفردية .
- الأسلوب **readlines()** يجعل من الممكن قراءة ملف كامل بتعليمة واحد فقط. هذا ليس دائما ممكنا، فمثلا لو لم يكن الملف كبيرا يمكن وضعه في متغير وهذا معناه في ذاكرة الوصول العشوائي للحاسوب، لذا فمن الضروري أن يكون الحجم كافيا. فإذا كنت في حاجة لمعالجة ملفات ضخمة، وتريد أستخدام الأسلوب **readline()** في حلقة، كما في المثال التالي.
- لاحظ أن **readline()** هو أسلوب يرجع سلسلة نصية، في حين أن **readlines()** ترجع قائمة. في نهاية الملف، **readline()** تقوم بإرجاع قناة فارغة، في حين أن **readlines()** تقوم بإرجاع سلسلة فارغة.

السكريبت التالي يوضح كيفية إنشاء دالة للتعامل مع الملفات النصية. في هذه الحالة، سوف يقوم باستنساخ الملف النصي وسيحذف جميع الأسطر التي تبدأ برمز '#' :

```
def filtre(source,destination):
    "recopier un fichier en éliminant les lignes de remarques"
    fs = open(source, 'r')
    fd = open(destination, 'w')
    while 1:
        txt = fs.readline()
        if txt =='':
            break
        if txt[0] != '#':
            fd.write(txt)
    fs.close()
    fd.close()
    return
```

لاستدعاء هذه الدالة، يجب عليك استخدام برامترين : اسم الملف الأصلي، اسم الملف الذي سيتلقى النسخة التي تمت تصفيتها. على سبيل المثال :

```
filtre('test.txt', 'test_f.txt')
```

تسجيل وعرض مختلف المتغيرات

البرامتر المستخدم مع الأسلوب **write()** في الملف النصي يجب أن يكون سلسلة نصية. مع الذي تعلمناه حتى الآن، نحن لا نستطيع حفظه مع أنواع أخرى من البيانات لذا لنسطيع حفظها في ملف يجب علينا أن نحولها إلى سلسلة نصية (string). يمكننا أن نفعل ذلك بمساعدة الدالة المدمجة **str()** :

```
>>> x = 52
>>> f.write(str(x))
```

سوف نرى لاحقا أنه يوجد طرق أخرى لتحويل قيم رقمية إلى سلاسل نصية (انظر : تنسيق السلاسل النصية، صفحة 149). لكن السؤال ليس هنا. إذا قمنا بحفظ قيم رقمية بتحويلها أولا إلى سلاسل نصية، سوف نستطيع إذن أن نعيد تحويلها إلى قيم رقمية عندما نقوم بقراءة الملف. على سبيل المثال :

```
>>> a = 5
>>> b = 2.83
>>> c = 67
>>> f = open('Monfichier', 'w')
>>> f.write(str(a))
>>> f.write(str(b))
>>> f.write(str(c))
>>> f.close()
>>> f = open('Monfichier', 'r')
>>> print(f.read())
52.8367
>>> f.close()
```

لقد قمنا بحفظ ثلاثة قيم رقمية. لكن كيف يمكننا التميز بين السلاسل النصية الناتجة، عندما نقوم بتشغيل الملف ؟ هذا مستحيل ! لا شيء يشير إلى أنه يوجد 3 قيم بدلا من واحدة أو 2 أو 4 ....إلخ

هنالك عدة حلول لهذه المشكلة. أفضلها هو استدعاء وحدة بيثون متخصصة : الوحدة **pickle**[49]. وهذا شرح لكيفية استخدامها :

```
>>> import pickle
>>> a, b, c = 27, 12.96, [5, 4.83, "René"]
>>> f =open('donnees_test', 'wb')
>>> pickle.dump(a, f)
>>> pickle.dump(b, f)
>>> pickle.dump(c, f)
>>> f.close()
>>> f = open('donnes_test', 'rb')
>>> j = pickle.load(f)
>>> k = pickle.load(f)
```

49 في اللغة الإنكليزية, المصطلح pickle معناه "حافظ". ولقد أطلق هذا الاسم على هذه الوحدة لأنها تستخدم لتخزين البيانات مع الحفاظ على نوعها .

```
>>> l = pickle.load(f)
>>> print(j, type(j))
27 <class 'int'>
>>> print(k, type(k))
12.96 <class 'float'>
>>> print(l, type(l))
[5, 4.83, 'René'] <class 'list'>
>>> f.close()
```

كما ترى في هذا المثال القصيرة، الوحدة pickly تسمح لك بحفظ البيانات مع المحافظة على نوعها. يتم تخزين محتويات المتغيرات الثلاثة **b**، **a** و **c** في ملف donnees_test، ومن ثم ننشئها مرة أخرى، مع نوعها في المتغيرات **k**، **j** و **l**. يمكنك إذا تخزين القيم من نوع "الصحيحة" و"سلاسل نصية" و"قوائم" والأنواع الأخرى التي سوف ندرسها في وت لاحق .

تنبيه، الملفات التي يتم معالجتها بمساعدة دالة الوحدة picle ليس بملفات نصية، لكنها ملفات ثنائية[50]. لهذا السبب، يجب أن تكون مفتوحة بمساعدة الدالة **open()** مثلا. ويجب علك استخدام البرامتر 'wb' لفتح ملف نصي والكتابة فيه (كما في السطر الثالث من مثالنا)، والبرامتر 'rb' لفتح الملف الثنائي والقراءة منه(مثل السطر الثامن من مثالنا).

الدالة **dump()** من الوحدة **pickle** تحتاج برامترين : الأول هو المتغير الذي سوف يتم حفظه، والثاني هو ملف الكائن الذي تعمل فيه. الدالة **pickle.load()** تنفذ عملها، وهذا معناه عودة كل متغير مع نوعه .

التعامل مع الاستثناءات: التعليمات try - except - else

الاستثناءات هي عمليات تعمل عندما يكشف المترجم أو المفسر خطأ أثناء تنفيذ البرنامج. عموما، يتم وقف تنفيذ البرنامج، ويتم إظهار رسالة خطأ أو أكثر. على سبيل المثال :

```
>>> print(55/0)
ZeroDivisionError: int division or modulo by zero
```

> يتم عرض معلومات أخرى، تشير على وجه الخصوص في البرنامج النصي على الخطأ الذي تم كشفه، ولكنها لا تتكاثر هنا .

رسالة الخطأ نفسها تحتوي على جزئين يفصل بينهما بنقطتين : الجزء الأول هو نوع الخطأ والجزء الثاني هو معلومات هذا الخطأ.

---

[50] في الإصدارات السابقة لبيثون، الوحدة pickle يتم استخدامها مع الملفات النصية (و لكن السلاسل النصية يتم معالجتها داخليا مع الاتفاقيات مختلفة). بيانات الملفات يتم صنعها مع إصدارات مختلفة لبيثون غير متوافقة مباشرة. المحولات موجودة.

في الكثير من الحالات، من الممكن التنبؤ مسبقا ببعض الأخطاء التي قد تحدث في أي لحظة معينة في البرنامج، وسوف نضم تعليمات محددة والتي يتم تفعيلها إذا حدثت هذه الأخطاء. في اللغات عالية المستوى مثل بيثون، من الممكن أيضا ربط آلية رصد مجموعة من التعليمات، وبالتالي تبسيط معالجة الأخطاء التي قد تحدث في أي من هذه التعليمات.

و تسمى الآلية من هذا النوع عامة "آليات التعامل مع الاستثناءات". بيثون تستخدم التعليمات **try - except - else**، التي تمكنك من التقاط الخطا وتشغيل جزء من سكريبت محدد لهذا الخطأ. وهي تعمل على النحو الأتي.

يتم تنفيذ كتلة البيانات مباشرة بعد التعليمة **try** فإذا حدث خطأ أثناء تنفيذ أي من هذه التعليمات، سيقوم بيثون بإلغاء التعليمة المخالفة ويتم تنفيذ بدل عنها كتلة التعليمات **except**. فإذا لم تقع أيت أخطاء يتم تنفيذ التعليمات التي تلي التعليمة **try**، وإذا الكتلة التي تلي التعليمة **else** يتم تنفيذهن (إذا كانت هذه التعليمة موجودة). في جميع الحالة، يستمر عمل البرنامج مع التعليمات الأخرى.

على سبيل المثال، انظر في السكريبت الذي يطلب من المستخدم إدخال اسم الملف الذي يريد قرائته. فإذا كان الملف غير موجود نحن لا نريد إيقاف البرنامج وإظهار الخطأ. ما نريده هو عرض التحذير وجعل المستخدم يدخل اسما آخر .

```
filename = input("Veuillez entrer un nom de fichier : ")
try:
    f = open(filename, "r")
except:
    print("Le fichier", filename, "est introuvable")
```

إذا كنا نرى أن تعمل مثل هذه التجارب في أكثر من مكان في البرنامج، يمكننا إدراجها في وظيفة :

```
def existe(fname):
    try:
        f = open(fname,'r')
        f.close()
        return 1
    except:
        return 0

filename = input("Veuillez entrer le nom du fichier : ")
if existe(filename):
    print("Ce fichier existe bel et bien.")
else:
    print("Le fichier", filename, "est introuvable.")
```

و من الممكن أيضا استخدام التعليمة **try** من مجموعة كتل **except**، كل واحدة منها تعمل عند نوع معين من الأخطاء، لكننا لن نطور ملاحق هنا. إذا أردت يرجى الرجوع إلى مراجع لغة بيثون .

تمارين

1.9 اكتب سكريبت يسمح بصناعة وقراءة ملف نصي. يجب على برنامج أن يطلب من المستخدم إدخال اسم الملف. ثم تظهر اختيار، إما بحفظ أسطر جديد أو إظهار محتوى الملف. يجب أن يدخل المستخدم الأسطر على التوالي من النص

باستخدام زر الإدخال، لفصل كل سطر عن الأخر. لإكمال المدخلات يجب على المستخدم إدخال سطر فارغ (و هذا معناه الضغط على زر الإدخال فقط). ويجب أن تظهر أسطر الملف مفصولة عن بعضها البعض بشكل طبيعي (الرمز الخاص بنهاية السطر يجب أن لا يظهر) .

2.9 نفترض أنك لديك ملف نصي يحتوي على جمل مختلفة الأطوال. اكتب برنامجا يعرض لك أطول جملة .

3.9 اكتب سكريبت يولد تلقائيا ملف نصي يحتوى على جداول الضرب من 2 إلى 30 (كل جدول يجب أن يتكون من 20) .

4.9 اكتب سكريبت يقوم بنسخ ملف نصي ويقوم بمضاعفة المسافات بين الكلمات ثلاثة مرات .

5.9 لديك تحت تصرفك ملف نصي في كل سطر يحتوي على قيمة عددية من نوع "حقيقي". على سبيل المثال :

```
14.896
7894.6
123.278
```

إلخ.

اكتب سكريبت يقوم بنسخ هذه القيم في ملف آخر ويقوم بتقريبها إلى أقرب عدد صحيح(التقريب يجب أن يكون صحيحا)

6.9 اكتب سكريبت يقوم بمقارنة محتويات ملفين ويظهر أو إختلاف يصادفه .

7.9 بداية نفترض أنك لديك ملف نصي يحتوي على جمل مختلفة الأطوال. اكتب برنامج الذي يعرض لك أطول جملة .من الملفين الموجودين مسبقا A و B، أصنع ملف C الذي يحتوي بالتناول على عنصر من A وعنصر من B إلى أن يصل إلى نهاية واحدة من الملفين الأصليين. قم بإكمال C من خلال لعناصر المتبقية في الملف الأخر .

8.9 اكتب برنامجا يقوم بتشفير ملف نصي يحتوي على أسماء، وألقاب وعناوين ورموز بريدية من أشخاص مختلفين(مثلا أعضاء في نادي) .

9.9 اكتب برنامجا يقوم بنسخ الملف المستخدم في التمرين السابق، ثم يقوم بإضافة تاريخ ولادة وجنس كل الأشخاص(سيقوم الحاسوب بإظهار الأسطر واحدة واحدة ويطلب من المستخدم إدخال البيانات الإضافية).

10.9 افترض أنك قمت بحل جميع التمارين السابقة وأصبح لديك الآن ملف يحتوي على معلومات العديد من الناس. اكتب سكريبت يقوم باستخراج الأسطر التي تحتوي على رموز البريدية من هذا الملف .

11.9 عدل السكريبت في التمرين السابق، بطريقة تجعله يجد الأسطر التي تحتوي على أسماء الأشخاص التي تبدأ أسمائهم F و M بطريقة أبجدية .

12.9 اكتب الدالات التي تؤدي نفس العمل وحدة **pickle** (انظر الصفحة 124). هذه الدالات يجب أن تسمح بحفظ المتغيرات في ملف، وترافقها معلومات دقيقة حول نوعها.

10

المزيد من هياكل البيانات

حتى الآن، لقد قمنا بعمليات بسيطة. سوف نتحرك الآن بالسرعة القصوى. إن هياكل البيانات التي قد استخدمتها حتى الآن لديها بعض الميزات التي لا تعرفها، وحان الوقت أيضا لأعرض عليكم هياكل غيرها أكثر تعقيدا .

النقطة على سلاسل النصية

لقد درسنا بالفعل السلاسل النصية في الفصل الخامس. وعلى عكس البيانات الرقمية، والتي هي كيانات فردية، السلاسل النصية من نوع بيانات المركبة. ونعني بذلك أنها متكونة من كيانات أصغر وهي : الحروف. تبعا للظروف، يجب علينا أن نتعامل مع هذه البيانات المركبة، أحيانا كائنا واحد، وفي بعض الأحيان تسلسل عناصر في الحالة الأخيرة، ربما نريد الوصول إلى كل عنصر من هذه العناصر على حدة.

في الحقيقة، السلاسل النصية هي جزء من فئة من كائنات بيثون تسمى المتسلسلات، وتنتمي إلى هذه الفئة القوائم والمصفوفات المغلقة (tuples). يمكننا القيام على المتسلسلات العمليات نفسها، ربما قد تكون تعرف بعضها، سوف نقوم بشرح عدد قليل منها في الفقرات القادمة .

Indiçage والاستخراج والطول

تذكير صغير بالفصل الخامس : السلاسل هي تسلسل من الحروف. كل واحدة منهم تأخذ مكانا خاصا في التسلسل. في بيثون، عناصر التسلسل دائما مفهرسة (أو مرقمة) بنفس الطريقة، هذا يعني أنها تبدأ من الصفر. لاستخراج حرف من السلسلة، يكفي أن تضع اسم المتغير الذي يحتوي على السلسلة ، ثم تضع مؤشره بين قوسين (قوسي نصف مربع أي : []) :

```
>>> nom = 'Cédric'
>>> print(nom[1], nom[3], nom[5])
é r c
```

غالبا يكون تحديد مكان حرف في نهاية السلسلة مفيدا. للقيام بذلك، يجب استخدام مؤشرات سالبة. مثلا 1- للحرف الأخير و 2- للحرف قبل الأخير ...إلخ :

```
>>> print (nom[-1], nom[-2], nom[-4], nom[-6])
cid
>>>
```

فإذا أردت تحديد عدد أحرف في السلسلة، نستخدم الدالة المدمجة **len()** :

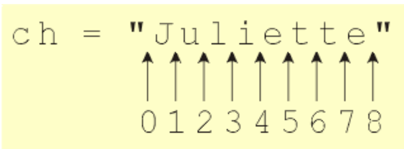
```
>>> print(len(nom))
6
```

استخراج أجزاء سلسلة

في الكثير من الأحيان، عندما تعمل مع السلاسل، قد ترغب بإسستخراج سلسلة صغيرة من سلسلة طويلة. بيثون يوفر لك تقنية بسيطة تسمى التقطيع ("التشريح"). وتتكون هذه التقنية من قوسين (نصف مربع : []) وفي داخلها مؤشرات بداية ونهاية الشريحة التي تريد استخراجها :

```
>>> ch = "Juliette"
>>> print(ch[0:3])
Jul
```

في شريحة **[n,m]** يتم تضمين بداية من n، ولا يتم تضمين m. فإذا أردت تخزين هذه الألية بسهولة، يجب أن تمثل أدلة تشير للمواقع في ما بين الحروف، كما في الرسم بالأسفل :



و نظرا لهذا النموذج، فإنه ليس صعبا أن تعرف أن استخراج **ch[3:7]** تساوي **"iett"**

مؤشرات الشريحة لديها قيم افتراضية : يعتبر أن أول مؤشر غير معرف مثل الصفر، في حين أن المؤشر الثاني يعتبر أن طول السلسلة الكاملة :

```
>>> print(ch[:3])        # أول ثلاثة أحرف
Jul
>>> print(ch[3:])        # كل ما بعد 3 أحرف
iette
```

و ينبغي على الحروف المعلمة أن لا تواجه مشاكل :

```
>>> ch = 'Adélaïde'
>>> print(ch[:3], ch[4:8])
Adé aïde
```

التسلسل، التكرار

و يمكن للسلاسل أن ترتبط بواسطة العامل **+** وأن تتكرر بواسطة العامل ***** :

```
>>> n = 'abc' + 'def'        # جمع سلسلة
>>> m = 'zut ! ' * 4         # تكرار
>>> print(n, m)
abcdef zut ! zut ! zut ! zut !
```

نلاحظ أن العاملين **+** و***** يستطيعان أيضا أن يستخدموا للجمع وضرب عند تطبيقها على برامترات رقمية. الحقيقة أن نفس العوامل يمكن أن تعمل بشكل مختلف تبعا للسياق الذي تعمل فيه وهي تستخدم الية مثيرة جدا تسمى حمولة الزائدة للمشغل. في لغات البرمجة الأخرى، الحمولة الزائدة للعوامل ليست ممكنة دائما : ويجب علينا استخدام رموز مختلفة للإضافة والتكرار، على سبيل المثال .

تمارين

1.10 أعرف بنفسك ماذا يحدث، عند تقطيع السلسلة، عندما يكون واحد أو أكثر من المؤشرات الشريحة خاطئة، وصف ذلك بأفضل طريقة ممكنة. (إذا كان المؤشر الثاني هو الأصغر من المؤشر الأول، على سبيل المثال' أو إذا كان المؤشر الثاني أكبر من حجم السلسلة) .

2.10 إقطع أجزاء سلسلة كبيرة على أجزاء كل واحدة بها 5 أحرف. ثم قم بجمع القطع في ترتيب عكسي. السلسلة يجب أن تحتوي على الأحرف المعلمة .

3.10 حاول كتابة دالة صغير اسمها **trouve()** التي تفعل بالضبط عكس الذي يفعله عامل المؤشر (هذا معناه ما بين قوسين [ ]). بدل من البدء بمؤشر ليرجع لك الحرف المطلوب، هذه الدالة تقوم بإرجاع مؤشر الكلمة التي تم إدخالها.

و بعبارة أخرى، اكتب دالة تأخذ برامترين : الأولى اسم السلسلة التي يجب معالجتها والثانية الحرف الذي يجب إيجاده. يجب على الدالة أن تقوم بإرجاع أول مؤشر للحرف في السلسلة. Ainsi par على سبيل المثال :

```
print(trouve("Juliette & Roméo", "&"))
```

يجب أن يطبع : **9**

انتبه : يجب أن تفكر في جميع الحالات المحتملة. وهذا يشمل أن تقوم الدالة بإرجاع قيمة معينة(على سبيل المثال -1) إذا كان الحرف الذي يبحث عنه غير موجود في السلسلة. ويجب على السلسلة أن تقبل الأحرف المعلمة .

4.10 حسن الدالة في التمرين السابق بإضافة برامتر ثالث : وهو مؤشر من أين يبدأ البحث في السلسلة. على سبيل المثال:

```
print(trouve ("César & Cléopâtre", "r", 5))
```

يجب أن تطبع : **15** (وليس 4 !).

5.10 اكتب الدالة **compteCar()** التي تحسب عدد مرات تكرار الحرف في السلسلة في السلسلة :

```
print(compteCar("ananas au jus","a"))
```

يجب أن تطبع : **4**

```
print(compteCar("Gédéon est déjà là","é"))
```

يجب أن تطبع : **3**

دورة من التسلسل : العبارة in ... for ...

كثيرا ما يحدث أنه يجب علينا أن نتعامل مع السلسلة النصية بأكملها بحرف، من الحرف الأول إلى الحرف الأخير للقيام بداية من أي واحدة أي معامل. ندعو هذه العملية بـ "دورة". سنقتصر فقط على أدوات بيثون التي نعرفها، نحن نستطيع أن نقوم بترميز هذه الدورة بمساعدة الحلقة، تتمحور حول العبارة **while** :

```
>>> nom ="Joséphine"
>>> index =0
>>> while index < len(nom):
...     print(nom[iindex] + ' *', end =' ')
...     index = index +1
...
J * o * s * é * p * h * i * n * e *
```

هذه الحلقة تقطع السلسلة **nom** لتستخرج كل حروفها واحدة واحدة، ثم تطبع معها علامة نجمة. انظر جيدا إلى الشرط المستخدم مع العبارة **while** وهو `index < len(nom)`، وهذا معناه أن الحلقة ستتوقف عندما تصل للمؤشر 9 (السلسلة تتكون من 10 حروف). وفي هذه الحالة سوف نتعامل مع كل حرف في السلسلة، لأن الفهرسة تبدأ من 0 وتنتهي بـ 9.

دورة التسلسل هي عملية متكررة جدا في البرمجة. لسهولة الكتابة، بيثون توفر لك هيكل حلقة أكثر ملائمة من الحلقة **while**، وهو الأسلوب **: ... in ... for**

مع هذه التعليمات، يصبح البرنامج في الأعلى هكذا :

```
>>> nom ="Cléopâtre"
>>> for car in nom:
...     print(car + ' *', end =' ')
...
C * l * é * o * p * â * t * r * e *
```

كما ترون، هيكل الحلقة هو الأكثر ملائمة. فهو يريحكم من تعريف وزيادة متغير معين (عداد) لإدارة مؤشر الحرف الذي تريدون معالجته في كل تكرار (بيثون هو الذي يفعل ذلك) الهيكل **in ... for ...** لا يظهر سوى الضروري، أي أن المتغير **car** سيحتوي على كل الحروف على التوالي، من البداية إلى النهاية.

العبارة **for** تتيح كتابة الحلقات، في تكرار يعالج فيه على التوالي كل عناصر لسلسلة المقدمة. في المثال أعلاه. كان التسلسل هو سلسلة نصية. المثال التالي يوضح أنه يمكن لنا أن نطبق نفس العملية على القوائم (و سيكون نفس الشيء مع ال tuples في وقت لاحق) :

```
liste = ['chien', 'chat', 'crocodile', 'éléphant']
for animal in liste:
    print('longueur de la chaîne', animal, '=', len(animal))
```

عند تشغيل هذا السكريبت يظهر لنا :

```
longueur de la chaîne chien = 5
longueur de la chaîne chat = 4
longueur de la chaîne crocodile = 9
longueur de la chaîne éléphant = 8
```

العبارة **for ... in ... :** هو مثال آخر من العبارة المجمعة. لا تنسَ النقطتين في نهاية السطر، ومسافة البادئة للكتلة التي تليها.

الاسم بعد الكلمة المحجوزة **in** هو الذي تجب معالجته. الاسم بعد الكلمة المحجوزة **for** هو اسم الذي اخترته لمتغير الذي سيحتوي على جميع عناصر التسلسل على التوالي. هذا المتغير سيتم تعريف تلقائيا (هذا معناه أنه ليس من الضروري أن تحدده سلفا، وسوف يتكيف تلقائيا لعنصر التسلسل التي تتم معالجته (تذكر أنه في حالة وجود قائمة، ليس بالضرورة أن تكون جميع عناصره من نفس النوع) على سبيل المثال :

```
divers = ['lézard', 3, 17.25, [5, 'Jean']]
for e in divers:
    print(e, type(e))
```

عند تشغيل هذا السكريبت فسوف يعطينا :

```
lézard <class 'str'>
3 <class 'int'>
17.25 <class 'float'>
[5, 'Jean'] <class 'list'>
```

على الرغم من أن عناصر القائمة مختلفة الأنواع (سلسلة نصية، عدد حقيقي، عدد صحيح، قائمة)، يمكننا أن نضعهم في المتغير **e**، دون ظهور أخطاء (وهذا أصبح ممكنا مع اختيار النوع التلقائي للمتغيرات في بيثون).

تمارين

6.10 في قصة أمريكية، تسمى 8 فراخ بط على التوالي : Pack، Oack، Nack، Mack، Lack، Kack، Jack و Qack.

اكتب سكريبت صغير يولد هذه الأسماء من السلسلتين التاليتين :

`prefixes = 'JKLMNOP'` et `suffixe = 'ack'`

إذا استخدمت العبارة **for ... in ...**، يجب على سكريبتك أن يتكون من سطرين فقط .

7.10 اكتب في سكريبت دالة تبحث عن عدد الكلمات التي تحتويها الجملة المقدمة .

8.10 اكتب سكريبت يبحث عن عدد الحروف e، é، è، ê، ë التي تحتويها الجملة المقدمة .

انتماء عنصر لتسلسل : استخدام التعليمة in وحدها

التعليمة **in** يمكن استخدامهابشكل مستقل عن **for**، للتحقق مما إذا كان العنصر المعين هو جزء من سلسلة أو لا. يمكنك على سبيل المثال استخدام التعليمة **in** للتحقق من أن حرفا أبجديا ما هو جزء من مجموعة معينة أو لا :

```
car = "e"
voyelles = "aeiouyAEIOUYàâéèêëùîï"
if car in voyelles:
    print(car, "est une voyelle")
```

بنفس الطريقة، يمكنك التحقق من انتماء عنصر لقائمة :

```
n = 5
premiers = [1, 2, 3, 5, 7, 11, 13, 17]
if n in premiers:
    print(n, "fait partie de notre liste de nombres premiers")
```

> هذه التعليمة قوية جدا لأنها حلقة حقيقية للتسلسل. كتمرين. اكتب التعليمات التي من شأنها أن تفعل ذلك باستخدام الحلقة التقليدية باستخدام العبارة *while*.

تمارين

9.10 اكتب الدالة **estUnChiffre()** التي تقوم بإرجاع "صحيح"، إذا كان البرامتر المرر عبارة عن رقم، وإلا سوف يقوم بإرجاع "خطأ". أختبر جميع أحرف سلسلة الدورة بمساعدة الحلقة **for**.

10.10 اكتب الدالة **estUneMaj()** التي تقوم بإرجاع "صحيح" إذا كان البرامتر المرر هو حرف كبير. حاول النظر إلى الأحرف الكبيرة المعلمة !

11.10 اكتب الدالة **chaineListe()** التي تقوم بتحويل جملة إلى سلسلة من الكلمات .

12.10 استخدم الدالات في التي تم تعريفهم في التمارين السابقة لكتابة سكريبت الذي يقوم باستخراج من نص جميع الكلمات التي تبدأ بحرف كبير .

13.10 استخدم الدالات التي تم تعريفها في التمارين السابقة لكتابة دالة التي تقوم بإرجاع رقم الحرف الكبير الموجود في جملة المقدمة بواسطة البرامتر .

السلاسل الغير قابلة للتعديل

لا يمكنك تحرير محتويات سلسلة موجودة. وبعبارة أخرى، لا يمكنك استخدام المعامل [ ] على الجانب الأيسر من عبارة التكليف. على سبيل المثال، حاول تشغيل البرنامج الصغير بالأسفل (الذي يبحث بالتخمين لاستبدال حرف في سلسلة) :

```
salut = 'bonjour à tous'
salut[0] = 'B'
print(salut)
```

النتيجة المتوقعة من المبرمج الذي كتب هذه التعليمات هي "Bonjour a tous" (مع B كبيرة). لكن على عكس التوقعات، هذا السكريبت يظهر خطأ "TypeError: 'str' object does not support item assignment". وهذا الخطأ سببه السطر الثاني من البرنامج. عندما تحاول استبدال حرف بحرف آخر من السلسلة، ولكن هذا غير مسموح به.

لكن هذا السكريبت يعمل جيدا :

```
salut = 'bonjour à tous'
salut = 'B' + salut[1:]
print salut
```

في هذا المثال، في الحقيقة، نحن لم نغير سلسلة **salut**. نحن صنعنا متغيرا جديدا، بنفس الاسم، في السطر الثاني من السكريبت .

السلاسل متشابهة

كل عوامل المقارنة التي تحدثنا عنها التي تتحكم في تدفق التعليمات (هذا معنا التعليمات **if ... elif ... else**) تعمل أيضا مع السلاسل النصية. وهذا قد يكون مفيدا لفرز الكلمات حسب الترتيب الأبجدي :

```
while True:
    mot = input("Entrez un mot quelconque : (<enter> pour terminer)")
    if mot =="":
        break
    if mot < "limonade":
        place = "précède"
    elif mot > "limonade":
        place = "suit"
    else:
        place = "se confond avec"
    print("Le mot", mot, place, "le mot 'limonade' dans l'ordre alphabétique")
```

هذه المقارنة ممكنة، وذلك لأن في كل معايير الترميز، الرموز الرقمية تمثل الحروف التي تم تعينها في ترتيب أبجدي، على الأقل بالنسبة للحروف الغير معلمة. في نظام الترميز ASCII، على سبيل المثال C=67 ،B=66 ،A=65 ... إلخ.

أرجو منك أن تفهم أنه لا يعمل سوى للكلمات التي بالصغير أو بالكبير، التي لا تحتوي على أي حروف معلمة. في الحقيقة، إذا كنت تعرف إن الحروف الكبيرة والصغيرة تستخدم مجموعة من الرموز المتميزة. أما بالنسبة إلى الحروف المعلمة، فقد رأيت أنها

يتم ترميزها في خارج مجموعات تتألف من ASCII. إن بناء خوارزمية تقوم بفرز حسب الحروف الأبجدية والتي ترى في كل مرة حالة الحروف ولهجاتهم، ليست بالمهمة السهلة !

المعيار يونيكود

من المفيد في هذا المستوى الاهتمام بقيم المعرفات الرقمية المرتبطة بكل حرف، في بيثون 3 تكون مجموع الحروف (من صنف سلسلة حروف) هي سلاسل تخضع لنفس الترميز الموحد[51]، وهذا يعني بأن المحددات الرقمية لحروفها تكون أحادية المعنى (فلا يمكن أن يوجد إلا حرف مطبعي بالنسبة لكل رمز) وعالمية (فالمحددات المختارة تشمل مجموع الحروف المستعملة في مختلف لغات العالم).

و هذا يعني بأن المحددات الرقمية لحروفها تكون أحادية المعنى (فلا يمكن أن يوجد إلا حرف مطبعي بالنسبة لكل رمز) وعالمية (فالمحددات المختارة تشمل مجموع الحروف المستعملة في مختلف لغات العالم).

ففي بداية ظهور تكنولوجيا المعلوميات،في وقت كانت فيه قدرات تخزين أجهزة الحواسيب جد محدودة، بحيث لم نتخيل بأن هذه سوف تستعمل في يوم ما لمعالجة نصوص أخرى غير تلك المتعلقة بتقنية الاتصالات، وخاصة باللغة الإنجليزية. لذلك كان يبدو من المعقول أن نضع لهاته الحواسيب لغة تتكون من مجموعة حروف محدودة، وبهذا الشكل يمكننا أن نمثل كل واحدة من هاته الحروف بعدد قليل من البتات، وبهذا شغل أقل حيز ممكن في وحدات التخزين المكلفة في ذلك الوقت. فنظام الحروف *ASCII*[52] الذي اختير في ذلك الوقت كان يعتقد بأن 128 حرف كافية له (مع علمنا بعدد التركيبات الممكنة لمجموعات تتكون من 7 بتات[53]). وبتوسيعها بعد ذلك إلى 256 حرف ، أصبح من الممكن جعلها مناسبة لمتطلبات معالجة النصوص المكتوبة في لغات أخرى غير الإنجليزية، لكن وكثمن لذلك تشتتت المعايير (فعلى سبيل المثال ،معيار ISO-8859-1 (latin-1) يقوم بترميز جميع الحروف المعلمة في اللغة الفرنسية أو الألمانية (من بين لغات أخرى)، لكنها لا تستطيع ترميز أي حرف إغريقي عبري أو سريالي. فالنسبة لهاته اللغات، ينبغي على التوالي استعمال المعايير ISO-8859-5 ،ISO-8859-8، ISO-8859-7، والتي هي بالطبع غير متوافقة فيما بينها هذا إضافة إلى أنه وجب استعمال معايير أخرى مختلفة للغات مثل العربية أو التشيكية أو الغنهارية...

---

51 في الإصدارات السابقة لبيثون، السلاسل النصية من نوع string هي في الواقع تسلسل من البايتات (و التي تمثل أحرف، لكن مع عدد من القيود المزعجة نوعا ما). ولكن كان هنالك بالفعل نوع ثاني من السلاسل، وهو النوع Unicode لمعالجة السلاسل النصية بالمعنى الذي نفهمه نحن .

52 ASCII = *American Standard Code for Information Interchange*

53 في الواقع، لقد قمنا بإستخدام البايتات في ذلك الوقت، لكن واحدةً من بتات (جمع كلمة بت – bit) من البايت يجب أن يتم حفظها كـ"بت" التحكم لأنظمة اكتشاف الأخطاء. تحسنت هذه الأنظمة لاحقا بحيث صارت تسمح بتحرير البت الثامن لتخزين معلومات مفيدة: هذا يسمح بتمديد ASCII إلى 256 حرف (معايير ISO-8859 و إلخ ...)

وتكمن أهمية هاته المعايير القديمة في بساطتها، فهي تسمح بالطبع لمطوري تطبيقات الحواسيب اعتبار أن كل حرف مطبعي يمثل واحد بايت، وكنتيجة لذلك فإن مجموعة من الحروف لا تمثل سوى متوالية من بايتات.حيث أنها هي الطريقة التي اشتغل بها النموذج القديم للمعطيات من نوع سلسلة نصية في بيثون (في النسخ السابقة للنسخة 3.0).

ولكن، وكما أشرنا إلى ذلك بشكل موجز في الفصل 5، فلا يمكن أبدا لتطبيقات الحواسيب الحديثة أن تكتفي فقط بهاته المعايير الضيقة. فينبغي الآن أن نكون قادرين على ترميز جميع حروف أبجدية أي لغة في نفس النص. لذلك تم إنشاء تنظيم دولي يدعى كونسورتيوم يونيكود، الذي تمكن من تطوير معيار عالمي تحت اسم يونيكود، هذا المعيار الجديد يهدف إلى إعطاء كل حرف من كل نظام لغوي مكتوب اسما ومحدداً رقمياً، وذلك بطريقة موحدة، كيفما كان الحاسوب أو البرنامج المستعمل.

لكن هنا تطرح مشكلة، فبسعيه نحو الكونية، ينبغي على معيار يونيكود أن يعطي محدداً رقمياً مختلفاً للعديد من عشرات الالاف من الحروف. فبالطبع لا يمكن لكل هاته المحددات أن يتم ترميزها تحت أوكت واحدة. ولعله سيكون من المغري أن نعلن على أنه في المستقبل، سيكون بالإمكان ترميزه كل حرف باستعمال اثنين بايت (هذا الذي يعطينا 65536 إمكانية) أو باستعمال ثلاث (16 777 216 إمكانية) أو أربع (أكثر من أربع مليارات إمكانية). فكل واحدة من هاته الاختيارات الصعبة، ومع ذلك تنتج الكثير من السلبيات. أول هاته السلبيات وهي مشتركة بين الجميع، هي أننا وباستعمالنا لهاته الاختيارات نفقد التوافق مع العديد من الوثائق الحاسوبية الموجودة مسبقا، (وخصوصا البرامج)، التي استعمل في ترميزها المعايير القديمة، والتي هي مبنية على أساس نموذج (كل حرف يساوي 1 بايت)؛ثاني هاته السلبيات تكمن في عدم القدرة على تلبية مطلبين متناقضين: فإذا قبلنا باستعمال 2 بايت فنحن عندئذ نجازف بعدم إيجاد إمكانيات من أجل تعريف أحرف نادرة، أو سمات أحرف ستكون مطلوبة بالتأكيد في المستقبل؛ ومن ناحية أخرى، فإذا افترضنا استعمال ثلاث، أو أربع أو أكثر من ذلك، فنحن عندئذ نتجه نحو إهدار للموارد وبشكل وحشي، حيث أن أغلب النصوص المستعملة لا تحتاج إلا لعدد محدود من الأحرف وبذلك فالعدد الأكبر من هاته الأوكت لن يحتوي سوى على أصفار.

ولكي لا نجد أنفسنا محاصرون في قيود من هذا النوع، فإن معيار يونيكود لا يقوم بوضع أية قواعد تخص عدد البايتات أو البتات المحفوظة لاستعمالها في الترميز. حيث إن هذا المعيار يقوم بتحديد قيمة المعرف الرقمي المقترن مع كل حرف. حسب الحاجة، حيث أن كل نظام كمبيوتر هو حر في استعمال نظام الترميز الداخلي الذي يناسبه في ترميز هذا المعرف حسب الحاجة، كمثال على ذلك فيمكن ترميزه على شكل عدد صحيح عادي. وذلك كمعظم لغات البرمجة الحديثة، كلغة بيثون التي قد تم تجهيزها ببيانات من نوع حروف (أو سلسلة محارف)، والذي تتوافق تماما مع معيار يونيكود. فالتمثيل الداخلي لهاته الرموز الرقمية المقابلة ليس بذي أهمية للمبرمج.

سنرى لاحقا في هذا الفصل أنه من الممكن وضع في سلسلة من هذا النوع أي تركيبة أحرف من الأبجديات المختلفة (قد تكون ASCII قياسية، أحرف معلمة، رموز رياضية أو الأحرف اليونانية، السيريلية أو العربية، وما إلى ذلك.)، والتي يمكن تمثيلها داخليا بواسطة رمز رقمي فريد من نوعه.

تسلسل الأوكت : نوع البايت

في هاته المرحلة من التفسيرات الخاصة بنا، فنحن بحاجة ماسة لتوضيح شيء آخر...

لقد راينا سابقاً أن معيار يونيكود لا يقوم بتثبيت أي شيء آخر غير القيم الرقمية، بالنسبة لكل المعرفات القياسية التي مناط بها وبشكل لا لبس فيه مهمة وصف حروف الهجاء المكونة للغات العالم بأسره (أكثر من 240،000 لغة في نوفمبر تشرين الثاني 2005). ولكن ورغم ذلك فمعيار يونيكود لا يحدد بأي شكل من الأشكال كيف سيتم ترميز هاته القيم بشكل ملموس كمجموعة أوكتيات أو بايتات.

بالنسبة للعمل الداخلي للتطبيقات الحاسوبية، فهذا ليس بذي أهمية، فمصممي لغات البرمجة، مترجميها أو مفسريها سوف يتمكنون من الاختيار وبكامل حرية تمثيل هاته الحروف باستعمال 8، 16، 24، 32، 64 بايت، أو حتى تمثيلها باستعمال أعداد حقيقية ذات فاصلة العائمة: (على الرغم من أننا لا نرى الجدوى من ذلك)، هذا يبقى اختيارهم وهو لا يعنينا. لذلك ليس علينا القلق بشأن الشكل الفعلي للأحرف داخل سلسلة المحارف في بيثون.

ولكن وعلى عكس ذلك لوحدات الإدخال والإخراج. فبالنسبة لنا كمطورين فهو واجب علينا أن نبين وبشكل دقيق ما هو نوع المعطيات التي تنتظرها برامجنا، فهل هاته البيانات سيتم إدخالها باستعمال لوحة المفاتيح أو سيتم استيرادها من أي مصدر كيف ما كان.إضافة إلى ذلك وجب علينا اختيار شكل البيانات التي سيتم تصديرها إلي جهاز محيط، سواء أكان طابعة، قرص صلب، أو شاشة.

فبالنسبة لوحدات الإدخال والإخراج الخاصة بالحروف، وجب علينا دائما أن نأخذ في الحسبان، أن الأمر يتعلق فعليا بمتوالية من الأوكتيات، وأنه يجب استخدام آليات مختلفة لتحويل سلاسل الأوكتيات هاته إلى سلاسل حروف والعكس بالعكس.

بيثون يتيح اليوم نوعا جديدا من البيانات يدعى (البايت)، وقد تم تطويره على وجه التحديد من أجل التعامل مع متواليات (أو سلاسل) الأوكت. فالبيانات من نوع بايت تشبه كثيرا البيانات من نوع سلسلة المحارف، مع فارق بسيط هو أنها تعتبر متواليات أوكتيات، وليس تسلسل لأحرف. ولكن من المؤكد أن الأحرف يمكن أن يتم ترميزها على شكل أوكت، والأوكت يفك تشفيرها لتغدو حروفا، ولكن ليس بشكل لا لبس فيه: حيث أن هناك معايير عدة لترميز وفك التشفير، لنفس السلسلة التي يمكن تحويلها إلى عدة سلاسل من البايتات المختلفة.

فعلى سبيل المثال[54]، سوف نقوم بتمرين بسيط حول الكتابة والقراءة في ملف نصي باستعمال سطر الأوامر، مستغلين بعضا من الإمكانيات التي توفرها الدالة **open()** والتي لم نصادفها حتى الآن، حيث سنسعى للقيام بهذا التمرين مع سلسلة تحتوي على عدد قليل من الحروف المعلمة، ورموز أخرى غير.

```
>>> chaine = "Amélie et Eugène\n"
>>> of =open("test.txt", "w")
>>> of.write(chaine)
17
>>> of.close()
```

مع هذه الأسطر القليلة، لقد قمنا بحفظ سلسلة نصية في هيئة أسطر نصية في ملف، بالطريقة المعتادة. دعونا نقوم بقراءة هذا الملف، لكن سوف نقوم بفتحه في الوضع البيناري (الثنائي)، وسوف نقوم بتمرير البرامتر "rb" إلى الدالة **open()**. في هذا الوضع، يتم نقل البايتات بوضع خام، دون تحويل من أي نوع. والقراءة عن طريق الدالة **read()** التي ستعطينا سلسلة نصية كما في الفصل السابق، لكن في سلسلة من البايتات، والمتغير الذي يتلقى تلقائيا نوع المتغير كنوع بايت :

```
>>> of =open("test.txt", "rb")          # "rb" => وضع القراءة (r) بيناري (b)
>>> octets =of.read()
>>> of.close()
>>> type(octets)
<class 'bytes'>
```

بذلك، نحن لن نسترد السلسلة الأصلية، ولكن في ترجمته بالبايت. حاول عرض هذه المعطيات بمساعدة الدالة **print()** :

```
>>> print(octets)
b'Am\xc3\xa9lie et Eug\xc3\xa8ne\n'
```

ماذا تعني هذه النتيجة ؟ عندما نطلب من بيثون عرض معطيات من نوع بايت بمساعدة الدالة **print()**، بيثون يوفر لنا في الواقع التمثيل، بين علامتي اقتباس للإشارة إلى أنها سلسلة، ولكن هذه سبقت بحرف **b** صغيرة الخاصة التي تعيين سلسلة من نوع بايت، مع هذه المصطلحات :

- و تتمثل قيم البايت الرقمية بين 32 و 127 من حروف ال ASCII.
- يتم تمثيل بعض القيم الرقمية التي تقل عن 32 بطرق تقليدية، مثل رمز (أو حرف) نهاية السطر.
- أما البايتات المتبقية فتتمثل قيمتها بأعداد الست العشرية، وسبقها \x.

---

[54] على سبيل المثال. نفترض أن التميز الافتراضي القياسي على نظام تشغيلك هو Utf-8. فإذا كنت تستخدم نظام تشغيل قديم يستخدم معايير مثل CP437 و CP850 و CP1252 و Latin-1) ISO8859-1). فإن النتائج قد تختلف قليلا فيما يتعلق بالأعداد وقيم البايتات. ولكن يجب أن لا تكون لديك صعوبة في تفسر ما تحصل عليه .

في مثالنا، جميع الحروف الغير معلمة للسلسلة يستخدم لترميز كل واحد منها بمساعدة بايت واحد الموافق لجدول ASCII : ونحن نعرف ذلك. أما للحروف المعلمة، (التي لا توجد في الكود ASCII)، يتم ترميزها ببايتين : مثلا \xc3 و\xa9 للحرف \ é، \xc3 و\xa8 لـ è. هذا الشكل المعين للترميز هو المعيار UTF-8، الذ سوف نشرحه بتفاصيل أكثر في الصفحات القادمة.

التمثيل الذي تم الحصول بواسطة **print()** يساعدنا على التعرف على سلسلتنا الأولية، لكنها لا تبين لنا جيدا بما فيه الكفاية بالبايتات. لذا دعونا نجرب شيئا آخر. هل تعرف ان للمرء أن يفحص محتوى التسلسل، عنصرا بعد عنصر بمساعدة دورة الحلقة. لنرى ما يحدث هنا :

```
>>> for oct in octets:
...     print(oct, end =' ')
...
65 109 195 169 108 105 101 32 101 116 32 69 117 103 195 168 110 101 10
```

هذه المرة، نحن نرى بوضوح البايتات : نحن قمنا بإرجاع جميع القيم الرقمية، بالترقيم العشري.

الحروف المعلمة التي تم ترميزها ببايتين في المعيار UTF-8، الدالة **len()** لا ترجع لنا نفس القيمة للسلسلة النصية، ولمعادلتها يجب علينا ترميزها بـ UTF-8 في سلسلة بايتات :

```
>>> len(chaine)
17
>>> len(octets)
19
```

عمليات استخراج العناصر، والتقطيع، إلخ ... تعمل بطريقة مماثلة مع البيانات من نوع بايت ونوع string، على الرغم من أن النتائج مختلفة، بطبيعة الحال :

```
>>> print(chaine[2], chaine[12], "---", chaine[2:12])
é g --- élie et Eu
>>> print(octets[2], octets[12], "---", octets[2:12])
195 117 --- b'\xc3\xa9lie et E'
```

انتبه، لا يمكن حفظ سلسلة من البايتات في ملف نصي. على سبيل المثال :

```
>>> of =open("test.txt", "w")
>>> of.write(octets)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not bytes
```

لحفظ سلسلة من البايتات، يجب عليك دائما فتح الملف في الوضع البيناري (الثنائي)، واستخدام البرامتر **"wb"** بدلًا من **"w"** في التعليمة **open()**.

ملاحظة أخيرة : نحن نستطيع أن نعرف متغيرا من نوع بايت وأن نعطيه قيمة حرفية، باستخدام هيكل جملة التالية :

`var = b'`*chaîne_de_caractères_strictement_ASCII*`'.`

الترميز UTF-8

كل ما سبق يشير إلى أن السلسلة النصية الأولية في مثالنا يتم تحويلها تلقائيا، عند حفظها في ملف، السلسلة من البايتات وفقا لمعايير ال UTF-8. التسلسل من البايتات التي ناقشناها حتى الآن تتوافق مع شكل معين من أشكال الترميز الرقمية، لسلسلة النصية "Amélie et Eugène ".

قد يبدو لك هذا للوهلة الأولى معقدا بعض الشيء، وانت تقول أن لسوء الحظ الترميز المثالي لا وجود له. اعتمادا على الذي سنقوم به، قد يكون من الأفضل ترميز النص بعدة طرق مختلفة. ولهذا السبب قد تم تعريفه، بالتوازي مع معايير Unicode، عدة معيايير ترميز : UTF-8 و UTF-16 و UTF-32 وبضعة خيارات أخرى. كل هذه المعايير تستخدم نفس المعرفات الرقمية لترميز الحروف، لكنهم يختلفون في طريقة حفظ هذه المعرفات في شكل بايتات. لا تقلق : على الأرجح أنك لن تتعامل سوى مع الأولى (UTF-8). الأخرون لا يقلقون سوى من المتخصصين في المجالات الأخرى.

معيار الترميز القياسي UTF-8 هو المعيار المفضل لمعظم النصوص الحالية، وذلك لأن :

- من ناحية إنه يتصمن توافقا تاما مع ترميز النصوص ASCII (كما هو الحال مع الكودات المصدرية للبرامج)، بالإضافة إلى أنه يتوافق جزئيا مع نصوص تم ترميزها مع مشتقاته، مثل Latin-1.
- من ناحية أخرى، هذا المعيار الجديد هو واحد من الأكثر كفائة في استخدام موارد الحاسوب، على الأقل النصوص المكتوبة بلغات غربية .

في هذا المعيار، يتم ترميز حروف ASCII القياسية في بايت واحد. أما بالنسبة للبقية فيتم ترميزها عادة في بايتين، وفي بعض الأحيان 3 أو 4 بايت للحروف الأكثر ندرة.

و على سبيل المقارنة، نذكر أن المعيار الأكثر استخداما من قبل الفرنكوفنيين قبل UTF-8 هو المعيار Latin-1 (و هو لا يزال واسع الانتشار، خاصة في بيئات عمل ويندوز تحت اسم CP1252[55]). هذا المعيار يسمح بترميز حرف بايت واحد لمجموعة معينة من الأحرف المعلمة، الموافق للغات الرئيسية لأوروبا الغربية (الفرنسية، الألمانية، البرتغالية، إلخ).

المعيارين UTF-16 و UTF-32 يتم الترميز فيهما ببايتين بالنسبة للأولى و 4 بايتات بالنسبة للثانية. لا تستخدم هذه المعايير سوى للاستخدامات الخاصة جدا، مثل معالجة السلاسل النصية الداخلية بواسطة مترجم أو مفسر .

[55] في نافذة موجه الأوامر في دوس في ويندوز إكس بي, الترميز الافتراضي هو CP850 .

التحويل (ترميز\فك ترميز) السلاسل

مع إصدارات بيثون التي سبقت الإصدار 3، مثل العديد من لغات البرمجة، فإنه في كثير من الأحيان يتم تحويل ترميز السلاسل النصية من معيار ترميز إلى آخر. لأن الاتفاقيات والاليات معتمد عليها الآن، ولن يكون لديك قلق على برامجك الخاصة للتعامل مع المعطيات الحديثة.

عندما حدث ذلك يجب علينا تحويل الملفات التي تم ترميزها وفقا لمعايير قديمة أو\و خارجية : المبرمج الذي يستحق هذا الاسم يجب أن يكون قادرا على أداء هذه التحويلات. لحسن الحظ، بيثون توفر لك الأدوات اللازمة، في شكل أساليب للكائنات المعنية .

تحويل سلسلة بايت إلى سلسلة نصية (string)

على سبيل المثال، انظر إلى تسلسل البايتات التي تم الحصول عليها في نهاية التمرين الصغير السابق. فإذا كنت تعلم فإن هذا التسلسل يتوافق مع النص وفقا لمعايير الترميز UTF-8، نحن نستطيع فك ترميز سلسلة نصية بمساعدة الأسلوب **decode()**، مع البرامتر **"Utf-8"** (أو بواسطة **"Utf8"**، **"utf-8"** أو **"utf8"**) :

```
>>> ch_car = octets.decode("utf8")
>>> ch_car
'Amélie et Eugène\n'
>>> type(ch_car)
<class 'str'>
```

هذا المسار للسلسلة المتحصل عليها يوفر لنا العديد من الحروف، وهذه المرة :

```
>>> for c in ch_car:
...     print(c, end =' ')
...
A m é l i e   e t   E u g è n e
```

تحويل سلسلة نصية (string) إلى سلسلة بايت

لتحويل سلسلة نصية إلى سلسلة بايت، يتم ترميزها وفقا للمعايير معينة، نستخدم الأسلوب **encode()**، والذي يعمل بشكل مماثل للأسلوب **decode()** المذكور أعلاه. على سبيل المثال، قم بتحويل نفس السلسلة النصية، إلى Utf-8 و Latin-1 للمقارنة بينهما.

```
>>> chaine = "Bonne fête de Noël"
>>> octets_u = chaine.encode("Utf-8")
>>> octets_l = chaine.encode("Latin-1")
>>> octets_u
b'Bonne f\xc3\xaate de No\xc3\xabl'
>>> octets_l
b'Bonne f\xeate de No\xebl'
```

في سلاسل بايت التي تم الحصول عليها، فمن الواضح أن الحروف المعلمة ê و ë تم ترميزها بمساعدة بايتين في حالة Utf-8، وبمساعدة بايت واحد في حالة Latin-1 .

التحويلات التلقائية عند معالجة الملفات

يجب عليك الآن إعادة النظر إلى ما يحدث عندما تريد تخزين سلسلة نصية في ملف نصي.

في الحقيقة حتى الآن، نحن لم نلفت الانتباه إلى مشكلة ترميز معايير هذه السلاسل، لأن الدالة **open()** لبيثون لديها لحسن الحظ إعدادات افتراضية مناسبة لحالات محددة حديثة. عند فتح ملف للكتابة عليه، على سبيل المثال، نحن نختار "w" أو "a" كباريمتر ثاني لـ **open()**، بيثون يقوم تلقائيا بترميز السلاسل بمعايير الافتراضية لنظام التشغيل الخاص بك (في أمثلتنا، يقوم بترميزها وفق معايير Utf-8)، ويتم تنفذ عمليات التحويل العكسي من خلال عمليات القراءة[56]. إذا، يمكننا اتباع منهج دراسة الملفات، في الفصل السابق، دون أن يعيقكم الشرح المفصل للغاية ..

في التمارين في الصفحات السابقة، نحن نواصل الاستغلال دون أن نقول أن هذه الإمكانيات توفرها بيثون. لكن انظر الآن كيفية حفظ النصوص من خلال تطبيق ترميز نصوص مختلفة عن تلك التي تقدم افتراضيا، لن يكون ذلك سوى لضمان أن الترميز الذي نريده (يجب علينا المضي قدما إذا كنا نريد أن تعمل سكريبتاتنا على أنظمة تشغيل مختلفة).

التقنية بسيطة. تشير فقط الترميز إلى **open()** بمساعدة برامترات إضافية: encoding ="*المعيار الذي اخترته*". À titre على سبيل المثال، يمكننا أن نكرر هذه التمارين من الصفحات السابقة ولكن يجب علينا هذه المرة استخدام الترميز Latin-1 ::

```
>>> chaine ="Amélie et Eugène\n"
>>> of =open("test.txt", "w", encoding ="Latin-1")
>>> of.write(chaine)
17
>>> of.close()
>>> of =open("test.txt", "rb")
>>> octets =of.read()
>>> of.close()
>>> print(octets)
b'Am\xe9lie et Eug\xe8ne\n'
```

... إلخ.

> يمكنك تنفيذ اختبارات مختلفة على هذه السلسلة من البيتات. إذا كنت ترغب في ذلك .

نفس الشيء عندما نفتح ملف للقراءة. افتراضيا، يقوم بيثون بفتح الملف وفقا للمعيار الافتراضي لنظام التشغيل. لكن هذا غير واضح من أنه مؤكد. على سبيل المثال دعونا نعيد فتح الملف (بدون مبالاة) test.txt الذي قمنا بصنعه في الخطوات السابقة :

[56] في الإصدارات السابقة لبيثون، يجب دائما على السلاسل النصية أن يتم تحويلها إلى سلسلة من البايتات قبل حفظها. والنوع السابق string هو ما يعادل نوع bytes الحالي. ويتم تحويل أية ملفات تلقائية عند عمليات قراءة\كتابة الملفات .

```
>>> of =open("test.txt", "r")
>>> ch_lue =of.read()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib/python3.1/codecs.py", line 300, in decode
    (result, consumed) = self._buffer_decode(data, self.errors, final)
UnicodeDecodeError: 'utf8' codec can't decode bytes in position 2-4:
invalid data
```

رسالة الخطأ واضحة : تم افتراض أن الملف تم ترميزه بـ Utf-8، وبيثون لا يمكنه فك ترميزه[57]. سوف يعمل إذا قمنا بما يلي:

```
>>> of =open("test.txt", "r", encoding ="Latin-1")
>>> ch_lue =of.read()
>>> of.close()
>>> ch_lue
'Amélie et Eugène\n'
```

في السكريبتات المتقدمة، من المحتمل دائما تحديد الترميز المفترض للملفات معالجته، حتى لو كان الطلب من المستخدم هذه المعلومات، أو فكر في تجارب متقدمة كثيرا أو قليلا لتحديد تلقائيا نوع الترميز .

حالة سكريبتات بيثون

سكريبتات بيثون هي في حد ذاتها ملفات نصية، بالطبع. بناءً على تكوين برنامج التحرير الخاص بك، أو على نظام التشغيل الخاص بك، يمكن لهذه النصوص أن يتم ترميزها بمعايير مختلفة. بحيث يمكن لبيثون تفسيره بشكل صحيح، وينصح لك دائما أن تشمل شبه التعليق هذا (يجب أن يكون في السطر الأول أو الثاني) :

```
# -*- coding:Latin-1 -*-
```

أو :

```
# -*- coding:Utf-8 -*-
```

... هذا يدل على الترميز المستخدم فعليا، بطبيعة الحال !

و بالتالي مفسر بيثون يعرف كيف يفك السلاسل النصية التي استخدمتها في السكريبت. لاحظ أنه يمكنك حذف هذا الشبه تعليق إذا كنت متأكدا أنه يتم ترميز النصوص الخاص بك بترميز UTF-8، لأنه هو الآن المعيار الافتراضي لنصوص بيثون[58] .

---

[57] في المعلوماتية، نسمي codec (ترميز\فك ترميز) أي تحويل شكل. سوف ترى على سبيل المثال العديد من codecs (ترميزات) في عالم المعلوماتية (ترميز الصوت، الفيديو). وبيثون لديه العديد من برامج تحويل السلاسل النصية لتحويل السلاسل وفقا لمعايير مختلفة.

[58] في الإصدارات السابقة لبيثون، الترميز الافتراضي هو ASCII .

الوصول إلى حروف أخرى غير الموجودة على لوحة المفاتيح

دعونا نرى أي جزء يمكنك أن تستخرجه في الحقيقة جميع الحروف لديها معرف رقمي عالمي يونيكود. للوصول إلى هذه المعرفات، يوفر لك بيثون عددا من الدالات المعرفة مسبقا.

الدالة **ord(ch)** تقبل أي حرف كبرامتر. وترجع القيمة الرقمية للحرف. مثلا `ord("A")` تقوم بإرجاع العدد 65، و `ord("Ĩ")` تقوم بإرجاع العدد 296.

الدالة **chr(num)** تقوم بالعكس تماما، تعطيها الحرف المطبعي وهي تقوم بإرجاع معرف اليونيكود الذي يساوي الرقم. ولتعمل هذه، يتطلب هذا استيفاء شرطين :

- قيمة **num** يجب أن تكون لحرف موجود مسبقا (معرفات اليونيكود ليست مستمرة : بعض الرموز لا تتطابق مع أي حرف)
- يجب على حاسوبك أن يعرف الوصف الرسومي للحرف، أو، بطريقة أخرى، يعرف رسم هذا الحرف، وهذا يسمى " glyphe" - "حرف رسومي". أنظمة التشغيل الحديثة تحتوى على مكتبات كبيرة للحروف الرسومية، والتي ينبغي لها أن تعرض الآلاف على الشاشة .

على سبيل المثال `chr(65)` تقوم بإرجاع الحرف A، و `chr(1046)` يرجع الحرف السيريلي **Ж**.

يمكنك استخدام هذه الدالات المعرفة مسبقا للهو باستكشاف لعبة إظهار الأحرف على شاشة الحاسوب. يمكنك على سبيل المثال أن تقوم بإرجاع الحروف الصغيرة للأبجدية اليونانية، مع العلم أن الرموز المخصصة لها تتراوح ما بين 945 إلى 969. انظر للسكريبت بالأسفل :

```
s = ""                          # سلسلة فارغة
i = 945                         # أول كود
while i <= 969:                 # آخر كود
    s += chr(i)
    i = i + 1
print("Alphabet grec (minuscule) : ", s)
```

يجب عليه أن يظهر النتيجة التالية :

```
Alphabet grec (minuscule) : αβγδεζηθικλμνξοπρςστυφχψω
```

تمارين

14.10 اكتب سكريبت صغير يجب عليه أن يظهر جدول رموز ASCII. يجب على البرنامج أن يقوم بإظهار جميع الحروف والرموز. بداية من هذا حدد العلاقة الرقمية البسيطة بين كل حرف كبير وبين كل حرف صغير يقابله .

15.10 عدل السكريبت السابق لاستكشاف الرموز ما بين 128 و 256، حيث ستجد حروف معلمة (من بين أمور كثيرة). هل العلاقة العددية التي وجدتها في التمرين السابق ستبقى نفسها للحروف المعلمة الفرنسية ؟

16.10 بداية من هذه العلاقة، اكتب دالة تحول جميع الحروف الصغيرة إلى حروف كبيرة، والعكس بالعكس (مقدمة في الجملة كبرامتر).

17.10 اكتب سكريبت يقوم بنسخ ملف نصي باستبدال جميع الفراغات (المساحات الفارغة) بهذه المجموعة من الرموز -*-. الملف الذي ستنسخه يجب أن يكون ترميزه بمعيار Latin-1، والملف الهدف سيكون ترميزه Utf-8. اسم الملفين سيتم طلبهم في بداية السكريبت .

18.10 اكتب الدالة **voyelle(car)**، التي تقوم بإرجاع "صحيح" إذا كان الحرف في البرامتر فوايال .

19.10 اكتب الدالة **compteVoyelles(phrase)**، التي تقوم بإرجاع عدد الحروف الفوايال في الجملة المقدمة .

20.10 اكتشف نطاق (مدى) حروف اليونيكود الموجودة في حاسوبك، بمساعدة حلقة برمجية مشابهة للتي قمنا باستخدامها لإظهار الأبجدية اليونانية. واعثر على الرموز المقابلة للحروف السيريلية، واكتب سكريبت يظهرها بالكبير والصغير.

السلاسل هي كائنات

في الفصول السابقة، لقد تعرفت إلى العديد من الكائنات. وأنت تعرف أنه يمكننا أن نعمل على كائن بمساعدة الأساليب (هذا معناه الدالات المرتبطة بهذا الكائن).

في بيثون، السلاسل النصية هي كائنات. لذلك يمكننا القيام بمعالجة السلاسل النصية باستخدام الأساليب الملائمة. وفي ما يلي بعضها، لقد إخترنا الأكثر استخداما[59]:

- **split()** : تحويل سلسلة إلى قائمة. يمكننا اختيار الحرف الذي يفصلها (يتم وضعه كبرامتر)، فإذا لم نضع سيكون الفراغ(مساحة فراغة) هي الافتراضية :

```
>>> c2 ="Votez pour moi"
>>> a = c2.split()
>>> print(a)
['Votez', 'pour', 'moi']
>>> c4 ="Cet exemple, parmi d'autres, peut encore servir"
>>> c4.split(",")
['Cet exemple', " parmi d'autres", ' peut encore servir']
```

59هذه ما هي إلا أمثلة قليلة. وأغلب هذه الأساليب يمكن أن يتم استخدمها مع برامترات مختلفة لم نحدد جميعها هنا (على سبيل المثال بعض البرامترات لا تسمح بمعالجة سوى جزء من السلسلة). يمكنك الحصول على قائمة كاملة من جميع الأساليب المرتبطة بكائن لاستخدام الدالة المدمجة dir(). يرجى الرجوع إلى أي كتاب مرجعي (أو وثائق على الإنترنت لبيثون) إذا كنت تريد أن تعرف أكثر من ذلك .

• join(liste) : جمع قائمة نصية إلى واحدة (هذا الأسلوب يعمل عكس الأسلوب السابق). تنبيه : السلسلة التي نطبق عليها هذا الأسلوب ستكون بمثابة فاصلة (واحدة أو أكثر) ; البرامتر المرر سيكون قائمة نصية التي نريد جمعها :

```
>>> b =["Bête", "à", "manger", "du", "foin"]
> >> print(" ".join(b))
Bête à manger du foin
>>> print("---".join(b))
Bête---à---manger---du---foin
```

• find(sch) : في سلسلة **sch** البحث عن مكان كلمة:

```
>>> ch1 = "Cette leçon vaut bien un fromage, sans doute ?"
>>> ch2 = "fromage"
>>> print(ch1.find(ch2))
25
```

• count(sch) : في سلسلة **sch** عدد تكرار الكلمة :

```
>>> ch1 = "Le héron au long bec emmanché d'un long cou"
>>> ch2 = 'long'
>>> print(ch1.count(ch2))
2
```

• lower() : تحويل سلسلة إلى حروف صغيرة :

```
>>> ch = "CÉLIMÈNE est un prénom ancien"
>>> print(ch.lower())
célimène est un prénom ancien
```

• upper() : تحويل سلسلة إلى حروف كبيرة :

```
>>> ch = "Maître Jean-Noël Hébèrt"
>>> print(ch.upper())
MAÎTRE JEAN-NOËL HÉBÈRT
```

• title() : تحويل أول حرف في كل كلمة إلى حرف كبير (مثل العناوين الإنكليزية):

```
>>> ch ="albert rené élise véronique"
>>> print(ch.title())
Albert René Élise Véronique
```

• capitalize() : بديل الأسلوب السابق. يحول فقط أول حرف في السلسلة إلى حرف كبير :

```
>>> b3 = "quel beau temps, aujourd’hui !"
>>> print(b3.capitalize())
"Quel beau temps, aujourd’hui !"
```

• swapcase() : يحول جميع الحروف الكبيرة إلى صغيرة والعكس بالعكس :

```
>>> ch = "Le Lièvre Et La Tortue"
>>> print(ch.swapcase())
lE lIÈVRE eT lA tORTUE
```

• strip() : حذف الفراغات في بداية ونهاية السلسلة :

```
>>> ch = "   Monty Python   "
>>> ch.strip()
'Monty Python'
```

• **replace(c1, c2)** : استبدال جميع الحروف **c1** بحروف **c2** في السلسلة:

```
>>> ch8 = "Si ce n'est toi c'est donc ton frère"
>>> print(ch8.replace(" ","*"))
Si*ce*n'est*toi*c'est*donc*ton*frère
```

• **index(car)** : إيجاد مؤشر أول ظهور للحرف **car** في السلسلة:

```
>>> ch9 ="Portez ce vieux whisky au juge blond qui fume"
>>> print(ch9.index("w"))
16
```

في معظم هذه الأساليب، يكون من الممكن تحديد جزء ما يجب معالجته، وهذا يكون بإضافة برامترات إضافية. على سبيل المثال :

```
>>> print (ch9.index("e"))       # البحث من البداية السلسلة
4                                # والعثور على أول حرف 'e'
>>> print (ch9.index("e",5))     # إبدأ فقط من المؤشر 5
8                                # الثانية 'e' وجِد
>>> print (ch9.index("e",15))    # إبدأ البحث بداية من المؤشر 15
29                               # وجِد رابع 'e'
```

إلخ.

أرجو منك أن تفهم أنه لا يمكن وصف كل الأساليب المتاحة، والبرامترات الخاصة بها، في إطار هذه الدورة. فإذا أردت المزيد من المعلومات، فيجب عليك قراءة وثائق بيثون (Library reference)، أو مراجع جيدة .

دالات مدمجة

لجميع الأغراض العملية، تذكر أيضا أنه يمكننا أيضا أن نطبق على السلاسل عددا كبيرا من الدالات المدمجة في اللغة :

• **len(ch)** إرجاع طول السلسلة **ch**، أو بعبارة أخرى، عدد أحرفها .

• **float(ch)** تحويل السلسلة **ch** إلى رقم حقيقي (*float*) (و بطبيعة الحال فإن هذا لا يعمل إلا إذا كانت السلسلة رقما، حقيقي أو صحيح)

```
>>> a = float("12.36")        # انتبه : ليس الفاصلة العشرية
>>> print a + 5
17.36
```

• **int(ch)** يحول السلسلة **ch** إلى عدد صحيح (مع نفس القيود) :

```
>>> a = int("184")
>>> print a + 20
204
```

• **str(obj)** يحول أو (يمثل) كائن **obj** إلى سلسلة نصية. **obj** يمكن أن يكون معطيات من أي نوع :

```
>>>> a, b = 17, ["Émile", 7.65]
>>> ch =str(a) +" est un entier et " +str(b) +" est une liste."
>>> print(ch)
17 est un entier et ['Émile', 7.65] est une liste.
```

تنسيق السلاسل النصية

لإكمال هذه النظرة العامة على المميزات المرتبطة بالسلاسل النصية، يبدو من الحكمة أن أقدم لك تقنية معالجة أكثر قوة، تسمى "تنسيق السلاسل". هذا مفيد جدا في جميع الحالات التي تحتاج بناء سلسلة معقدة من عدة قطع، مثل قيم المتغيرات المختلفة.

على سبيل المثال، اكتب برنامج الذي يعالج لون ودرجة حرارة محلول مائي، في الكيمياء، يتم تخزين اللون في سلسلة نصية تدعى **coul**، ودرجة الحرارة في متغير من نوع حقيقي يدعى **temp**. ويجب على برنامجك أن يقوم ببناء سلسلة نصية من هذه البيانات، على سبيل المثال، جملة مثل هذه : "المحلول سيكون أحمر ودرجة الحرارته 12.7 درجة مئوية ".

يمكنك بناء هذه السلسلة بجمع القطع مع بعضها البعض بمساعدة المعامل التسلسل (الرمز +)، ولكن يجب عليك أيضا استخدام الدالة المدمجة **str()** لتحويل سلسلة نصية والقيمة الرقمية موجودة داخل متغير من نوع float (قم بالتمارين) .

يوفر لك بيثون إمكانيات أخرى. يمكنك صنع سلسلة ("patron" - الزعيم) التي تحتوي على أغلب النص الذي لم يتغير مع علامات المواقع المحددة (حقول) حيث تظهر عندما تريد محتويات المتغيرات. قم الآن بتطبيق الأسلوب **format()**ـ على هذه السلسلة، وسوف توفرها على شكل برامترات والكائنات المختلفة تحول إلى حروف ويتم إضافتها بدل العلامات. مثال أفضل من كل هذا :

```
>>> coul ="verte"
>>> temp =1.347 + 15.9
>>> ch ="La couleur est {} et la température vaut {} °C"
>>> print(ch.format(coul, temp))
La couleur est verte et la température vaut 17.247 °C
```

تستخدم العلامات المتكونة من الأقواس، التي تحتوي أو قد لا تحتوي على معلومات التنسيق :

- إذا كانت العلامات فارغة (في أبسط الحالات)، سوف يتحصل الأسلوب **format()** على برامترات التي ستكون بمثابة العلامات في السلسلة. وسوف يقوم بيثون إذا بتطبيق الدالة **str()** على كل من هذه البرامترات، وسوف يضفها إذا في السلسلة في مكان العلامات، في نفس الترتيب. البرامترات يمكن أن تكون أي كائن أو تعبير بيثون :

```
>>> pi =3.1416
>>> r =4.7
>>> ch ="L’aire d’un disque de rayon {} est égale à {}."
>>> print(ch.format(r, pi * r**2))
L’aire d’un disque de rayon 4.7 est égale à 69.397944.
```

• يمكن للعلامات أن تحتوي على أرقام متسلسلة (العد يبدأ من الرقم 0) لوصف بدقة البرامترات التي ستمرر لـ **format()** لتحل مكانها. هذه التقنية هي قيمة خاصة إذا كانت نفس البرامتر يجب عليه استبدال مجموعة من العلامات المحددة:

```
>>> phrase ="Le{0} chien{0} aboie{1} et le{0} chat{0} miaule{1}."
>>> print(phrase.format("", ""))
Le chien aboie et le chat miaule.
>>> print(phrase.format("s", "nt"))
Les chiens aboient et les chats miaulent.
```

• يمكن للعلامات أن تحتوي على معلومات تنسيق (بالإشتراك أو ليس مع تسلسل الأرقام). على سبيل المثال، يمكنك تحديد بدقة النتيجة النهائية أو إجبار استخدام الرموز العلمية أو تحديد عدد الأحرف، ... إلخ :

```
>>> ch ="L'aire d'un disque de rayon {} est égale à {:8.2f}."
>>> print(ch.format(r, pi * r**2))
L'aire d'un disque de rayon 4.7 est égale à    69.40.
>>> ch ="L'aire d'un disque de rayon {0} est égale à {1:6.2e}."
>>> print(ch.format(r, pi * r**2))
L'aire d'un disque de rayon 4.7 est égale à 6.94e+01.
```

في الاختبار الأول، النتيجة تم تنسيقها بطريق لتحمل 8 أحرف، رقمين منهم بعد الفاصل. في التجربة الثانية، تم عرض النتيجة في شكل علمي (**e+01** معناها 10^{01} x). ملاحظة : يتم تنفيذ التقريبات المحتملة بشكل صحيح .

الوصف الكامل لجميع إمكانيات التنسيق يجب أن تكون في العديد من الصفحات، وهذا أكبر من حجم نطاق الكتاب. فإذا كنت في حاجة لمعرفة المزيد حول التنسيق، فيجب عليك الاطلاع على وثائق لغة بيثون، أو الكتب الأكثر تخصصا. ملاحظة بسيطة : هذا التنسيق يسمح بإظهار بسهولة النتيجة الرقمية بالترقيم الثنائي (بيناري) أو الثماني أو الست العشري :

```
>>> n =789
>>> txt ="Le nombre {0:d} (décimal) vaut {0:x} en hexadécimal et {0:b} en binaire."
>>> print(txt.format(n))
Le nombre 789 (décimal) vaut 315 en hexadécimal et 1100010101 en binaire.
```

سلاسل التنسيق " القديمة "

إصدارات بيثون قبل الإصدار 3 كانت تستخدم تقنيات تنسيق مختلفة قليلا وأقل تطورا، لكن لا تزال صالحة للاستخدام. وأنصحك بأن تعتمد الطريقة التي ذكرناها في الفقرات السابقة. وسوف نشرح هنا باختصار الطريقة القديمة، لأنك قد تواجهها في سكريبتات الكثير من المبرمجين (و حتى في بعض أمثلتنا !). وتتكون في تنسيق السلسلة بجمع عنصرين بمساعدة العامل %. على يسار هذا العامل، السلسلة "الرئيسة" التي تحتوي على علامات تبدأ دائما بـ %، وفي اليمين (بين قوسين) أين يضع بيثون الكائن في السلسلة، بدلا من العلامات .

مثال:

```
>>> coul ="verte"
>>> temp = 1.347 + 15.9
>>> print ("La couleur est %s et la température vaut %s °C" % (coul, temp))
La couleur est verte et la température vaut 17.247 °C
```

العلامة s% تلعب نفس دور {} في الطريقة الجديدة. وتقبل أي كائن (سلسلة، عدد صحيح، عدد حقيقي، قائمة ...). ويمكنك استخدام علامات أخرى أكثر تقدما، مثل 8.2f% أو 6.2e% التي هي مثل {8.2f:} و{6.2e:} في الطريقة الجديدة. وهذا في أبسط الحالات، لكن اقتنع أن إمكانيات الصيغة الجديدة هي واسعة .

تمارين

21.10 اكتب سكريبت الذي يقوم بنسخ ملف نصي تم ترميزه بـ Latin-1 إلى Utf-8، ويجب أن تكون كل كلمة تبدأ بحرف كبير. سوف يقوم البرنامج بطلب أسماء الملفات من المستخدم. المعاملات القراءة والكتابة للملفات في وضع الوضع الملف النصي العادي .

22.10 بديل التمرين السابق : فعّل عمليات قراءة وكتابة الملفات في وضع الثنائي، والعمليات ترميز\فك ترميز سلاسل البايتات. بالإضافة، يجب عليك التعامل مع الأسطر بطريقة لاستبدال جميع الفراغات بمجموعة من 3 رموز -*-.

23.10 اكتب سكريبت الذي يقوم بحساب عدد الكلمات الموجودة في ملف نصي .

24.10 اكتب سكريبت الذي يقوم بنسخ ملف نصي مع دمج (مع السابقة) الأسطر التي لا تبدأ بحرف كبير .

25.10 لديك ملف يحتوي على قيم رقمية. إعتبر أن هذه القيم هي أقطار من سلسلة من الكرات. اكتب سكريبت يقوم باستخدام معطيات هذا الملف لصنع ملف آخر، منظمة في أسطر من النصوص بوضوح الخصائص الأخرى لهذه الكرات (مساحة الجزء ومساحة الخارجية والحجم)، في جمل مثل هذه :

```
Diam.  46.20 cm Section  1676.39 cm² Surf.  6705.54 cm² Vol.  51632.67 cm³
Diam. 120.00 cm Section 11309.73 cm² Surf. 45238.93 cm² Vol. 904778.68 cm³
Diam.   0.03 cm Section     0.00 cm² Surf.     0.00 cm² Vol.      0.00 cm³
Diam.  13.90 cm Section   151.75 cm² Surf.   606.99 cm² Vol.   1406.19 cm³
Diam.  88.80 cm Section  6193.21 cm² Surf. 24772.84 cm² Vol. 366638.04 cm³
```

إلخ.

26.10 لديك تحت تصرفك ملف نصي يحتوي على أسطر تمثل قيما رقمية من نوع حقيقي، دون أن تعرض (و يتم ترميزها كسلاسل نصية). اكتب سكريبت يقوم بنسخ هذه القيم في ملف آخر، وتقريبهم بحيث لا تحتوي بعد الفاصل أكثر من رقم واحد، هذا الرقم لا يمكن أن يكون سوى 0 أو 5 (التقريب يجب أن يكون صحيحا) .

النقطة في القوائم

لقد التقينا القوائم بالفعل عدة مرات، منذ تقديمه باختصار في الفصل 5. القوائم هي مجموعات مرتبة من الكائنات. مثل السلاسل النصية، القوائم هي مجموعة من جزء من نوع عام الذي سميه في بيثون التسلسل. مثل الحروف في السلسلة، الكائنات تم وضعها في القائمة من خلال الفهرس (رقم الذي يشير إلى مكان الكائن في التسلسل).

تعريف قائمة - الوصول إلى عناصرها

أنت تعرف بالفعل أنه يتم تحديد القائمة بمساعدة الأقواس المعقوفة (نصف مربع) :

```
>>> nombres = [5, 38, 10, 25]
>>> mots = ["jambon", "fromage", "confiture", "chocolat"]
>>> stuff = [5000, "Brigitte", 3.1416, ["Albert", "René", 1947]]
```

في المثال الأخير أعلاه، قمنا بتجميع عدد صحيح وسلسلة وعدد حقيقي وحتى قائمة، لتذكيركم بأنه يمكن وضع معطيات من أي نوع في القائمة، بما في ذلك القوائم والقواميس و tuples (سوف نناقشها في وقت لاحق).

للوصول إلى عناصر قائمة، سوف نستخدم نفس الطرق (رقم المؤشر والتقطيع إلى قطع) للوصول إلى الأحرف في سلسلة :

```
>>> print(nombres[2])
10
>>> print(nombres[1:3])
[38, 10]
>>> print(nombres[2:3])
[10]
>>> print(nombres[2:])
[10, 25]
>>> print(nombres[:2])
[5, 38]
>>> print(nombres[-1])
25
>>> print(nombres[-2])
10
```

و ينبغي أن تلفت الأمثلة المذكورة أعلاه انتباهكم إلى أن قطعة (شريحة) من القائمة هي دائما قائمة (حتى لو كانت القطعة (الشريحة) تحتوي على عنصر واحد، كما في المثال الثالث) إذاً يمكن للعنصر الواحد أن يحتوي على معطيات من أي نوع. وسوف نقوم باستكشاف هذه الميزة طوال هذه الأمثلة القادمة .

القوائم يمكن تغييرها

على عكس السلاسل النصية، القوائم هي تسلسل قابل للتغيير. وهذا سيسمح لنا لاحقا ببناء قوائم كبيرة الحجم، قطعةً قطعة، بطريقة ديناميكية (و هذا معناه بمساعدة أي خوارزمية). على سبيل المثال :

```
>>> nombres[0] = 17
>>> nombres
[17, 38, 10, 25]
```

في المثال أعلاه، قمنا باستبدال العنصر الأول من القائمة **nombres**، باستخدام المعامل [ ] (على يسار علامة المساواة.

فإذا كنت تريد الوصول إلى عنصر في قائمة داخل قائمة أخرى. يكفي أن تشير ببساطة إلى مؤشر بين قوسين (نصف مربع) :

```
>>> stuff[3][1] = "Isabelle"
>>> stuff
[5000, 'Brigitte', 3.1415999999999999, ['Albert', 'Isabelle', 1947]]
```

كما هو الحال بالنسبة لجميع التسلسلات، يجب علينا أن لا ننسى أن الترقيم يبدأ من الصفر. وبالتالي، في المثال أعلاه قمنا باستبدال العنصر رقم 1 في قائمة، والذي هو العنصر 3 في قائمة أخرى : سلسلة **stuff**.

القوائم هي كائنات

في بيثون، القوائم هي كائنات في حد ذاتها، ويمكنك إذًا تطبيق عدد من الأساليب الفعالة عليها بشكل خاص. وهذه بعضها :

```
>>> nombres = [17, 38, 10, 25, 72]
>>> nombres.sort()                    # قم بفرز القائمة
>>> nombres
[10, 17, 25, 38, 72]

>>> nombres.append(12)                # أضف عنصر إلى النهاية
>>> nombres
[10, 17, 25, 38, 72, 12]

>>> nombres.reverse()                 # اعكس ترتيب العناصر
>>> nombres
[12, 72, 38, 25, 17, 10]

>>> nombres.index(17)                 # جد مؤشر عنصر
4

>>> nombres.remove(38)                # إحذف عنصر
>>> nombres
[12, 72, 25, 17, 10]
```

و بالإضافة إلى هذه الأساليب، يوجد أيضا التعليمة المدمجة **del** الذي تسمح لك بحذف عنصر أو أكثر من خلال مؤشره (أو مؤشراتهم) :

```
>>> del nombres[2]
>>> nombres
[12, 72, 17, 10]
>>> del nombres[1:3]
>>> nombres
[12, 10]
```

لاحظ الفرق بين الأسلوب **remove()** والتعليمة **del** : **del** تعمل مع مؤشر أو شريحة المؤشر في حين أن **remove()** تبحث عن القيمة (فإذا كان يوجد العديد من العناصر بنفس القيمة يتم مسح الأولى فقط) .

تمارين

27.10 اكتب سكريبت يقوم بإنشاء قائمة من المربعات والمكعبات عددها 20 إلى 40 .

28.10 اكتب السكريبت يقوم تلقائيا بصنع قائمة من المنحنيات ذات زوايا من 0° إلى 90°، في خطوات من 5°. تنبيه : الدالة **sin()** لوحدة **math** تعتبر أن الزوايا بالراديان (360° = 2 π راديان) .

29.10 اكتب سكريبت يسمح بعرض أول 15 نتيجة لجدول الضرب على 2، 3، 4، 5، 7، 13، 17، 19 (هذه الأقام سوف يتم وضعها في بداية القائمة) في شكل جدول مشابه للجدول التالي :

```
2   4   6   8  10  12  14  16  18  20  22  24  26  28  30
3   6   9  12  15  18  21  24  27  30  33  36  39  42  45
5  10  15  20  25  30  35  40  45  50  55  60  65  70  75
```

إلخ.

30.10 انظر للقائمة التالية : ['Jean-Michel', 'Marc', 'Vanessa', 'Anne', 'Maximilien', 'Alexandre-Benoît', 'Louise']

اكتب سكريبت يقوم بعرض اسم من هذه الأسماء مع عدد الحروف التي يتكون منها .

31.10 لديك قائمة من أي الأعداد صحيحة، بعضها مكرر في أماكن مختلفة في القائمة. اكتب سكريبت يقوم بنسخ هذه القائمة في قائمة أخرى مع حذف التكرار (سيتم فرز القائمة النهائية) .

32.10 اكتب سكريبت يبحث عن الكلمة الأطول في الجملة المقدمة (يجب على المستخدم أن يدخل الجملة حسب اختياره)

33.10 اكتب سكريبت يقوم بعرض قائمة بكل أيام السنة من مخيلتك، والتي تبدأ بيوم الخميس. السكريبت الخاص بك يستخدم 3 قوائم : قائمة بأيام الأسبوع، قائمة بأسماء الأشهر، وقائمة بعدد الأيام لكل شهر(تجاهل السنة الكبيسة).

على سبيل المثال :

```
jeudi 1 janvier   vendredi 2 janvier   samedi 3 janvier   dimanche 4 janvier
```

و هكذا إلى يوم 31 ديسمبر (كانون الأول)

34.10 لديك ملف نصي يحتوي على أسماء التلاميذ. اكتب سكريبت يقوم بنسخة مرتبة من هذا الملف .

35.10 اكتب دالة تقوم بفرز قائمة. هذه الدالة لا يجب عليها أن تستخدم الأسلوب المدمج **sort()** الخاص ببيثون : لذا يجب عليك أن تقوم بنفسك بكتابة خوارزمية الفرز .

تقنيات تقطيع متقدم للتعديل على قائمة

كما لاحظنا للتو، يمكنك إضافة أو حذف عناصر في قائمة باستخدام التعليمة (**del**) والأسلوب (**append()**) المدمج. فإذا كان لايزال لديك فهم أساسي "للتقطيع إلى شرائح"، يمكنك إذا الحصول على نفس النتائج بمساعدة معامل واحد [ ]. استخدام هذا المعامل هو أكثر عرضة للتلف من التعليمات أو الأساليب المخصصة، لكنه يسمح بمزيد من المرونة :

إدخال عنصر أو أكثر في أي مكان في القائمة

```
>>> mots = ['jambon', 'fromage', 'confiture', 'chocolat']
>>> mots[2:2] =["miel"]
>>> mots
['jambon', 'fromage', 'miel', 'confiture', 'chocolat']

>>> mots[5:5] =['saucisson', 'ketchup']
```

```
>>> mots
['jambon', 'fromage', 'miel', 'confiture', 'chocolat', 'saucisson', 'ketchup']
```

لاستخدام هذه التقنية، يجب عليك أن تعرف هذه المميزات :

- إذا استخدمت المعامل [ ] على يسار علامة المساواة لإدراج أو حذف عنصر أو عناصر في قائمة، يجب عليك أن تشير إلى "الشريحة" في القائمة المستهدفة (و هذا معناه مؤشرين الذين جمعتهما باستخدام الرمز :)، وليس عنصر واحد في هذه القائمة .
- يجب على العنصر الذي على يمين علامة المساوات أن يكون قائمة. فإذا لم تدرج سوى عنصر واحد، يجب عليك إذا تقديمه بين معقوفين لتحويل أولا إلى سلسلة بعنصر واحد. لاحظ أن العنصر **mots[1]** ليس قائمة (هو سلسلة "fromage")، إذا العنصر **mots[1:3]** في واحدة.

سوف تفهم بشكل أفضل من خلال تحليل ما يلي :

إزالة \ استبدال عناصر

```
>>> mots[2:5] = []              # [] يدل على قائمة فارغة
>>> mots
['jambon','fromage','saucisson', 'ketchup']

>>> mots[1:3] = ['salade']
>>> mots
['jambon', 'salade', 'ketchup']

>>> mots[1:] = ['mayonnaise', 'poulet', 'tomate']
>>> mots
['jambon', 'mayonnaise', 'poulet', 'tomate']
```

- في السطر الأول من مثالنا، قمنا باستبدال الشريحة [2:5] بقائمة فارغة، والذي يتوافق مع الذي حذفناه.
- في السطر الرابع، قمنا باستبدال شريحة بعنصر واحد. لاحظ مرة أخرى أن هذا العنصر هو في حد ذاته "يعرض" على شكل قائمة.
- في السطر السابع، قمنا باستبدال الشريحة بها عنصران بأخرى بها

إنشاء قائمة من الأرقام بمساعدة الدالة range()

إذا كان يجب عليك التعامل مع سلاسل من الأرقام، يمكنك إنشاؤها بسهولة بمساعدة هذه الدالة المدمجة. فهي تقوم بإرجاع سلسلة من الأعداد الصحيحة[60] التي يمكنك استخدامها مباشرة، أو تحويلها إلى سلسلة عن طريق الدالة **list()**، أو تحويلها إلى tuple بمساعدة الدالة **tuple()** (سوف نقوم بشرح ال tuples في وقت لاحق) :

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

الدالة **range()** تقوم افتراضيا بتوليد سلسلة من الأعداد الصحيحة بشكل متزايد، ومختلفة بعدد واحد. فإذا استدعيت **range()** مع برامتر واحد، القائمة ستحتوي على عدد الأرقام القيم المساوية للبرامتر المقدم، لكنها تبدأ من الرقم صفر (و هذا معناه أن **range(n)** سوف تقوم بتوليد الأرقام من 0 إلى (n-1). لاحظ أن البرامتر المقدم لن يكون في السلسلة.

يمكننا استخدام **range()** مع برامترين أو 4 برامترات مفصولة بفواصل، لتوليد متواليات من أعداد أكثر تحديدا :

```
>>> list(range(5,13))
[5, 6, 7, 8, 9, 10, 11, 12]
>>> list(range(3,16,3))
[3, 6, 9, 12, 15]
```

فإذا كان لديك صعوبة في استيعاب المثال أعلاه، افترض أن **range()** تنتظر منك دائما 3 برامترات، والتي يمكن أن نسميها FROM و TO و STEP. و FROM هي القيمة الأولى لتوليدها، TO هي الأخيرة (أو بالأحرى الأخيرة مع واحد)، و STEP هي الخطوة للقفز من قيمة لأخرى. فإذا لم نضعها فإن البرامترين FROM و STEP ستكون قيمتهم افتراضية هي 0 و 1.

و يسمح بالبرامترات السلبية :

```
>>> list(range(10, -10, -3))
[10, 7, 4, 1, -2, -5, -8]
```

تكرار القائمة بمساعدة for وrange() وlen()

العبارة **for** هي العبارة المثالية لتكرار قائمة:

```
>>> prov = ['La','raison','du','plus','fort','est','toujours','la','meilleure']
>>> for mot in prov:
...     print(mot, end =' ')
...
La raison du plus fort est toujours la meilleure
```

فإذا كنت تريد تكرار مجموعة من الأعداد الصحيحة، الدالة **range()** ستكون مناسبة:

---

[60] إن **range()** تعطي في الواقع الوصول إلى المكرر (كائن بيثون مولد لتسلسلات). ولكن شرحها خارج إطار الذي وضعناه لهذه الكتاب. يرجى الرجوع إلى قائمة مراجع صفحة 13، أو وثائق بيثون على الإنترنت إذا كنت تريد التوضيح .

```
>>> for n in range(10, 18, 3):
...     print(n, n**2, n**3)
...
10 100 1000
13 169 2197
16 256 4096
```

من المريح الجمع بين الدالات **range()** و **len()** للحصول تلقائيا على كافة مؤشرات لتسلسل (قائمة أو سلسلة). على سبيل المثال :

```
fable = ['Maître','Corbeau','sur','un','arbre','perché']
for index in range(len(fable)):
    print(index, fable[index])
```

تشغيل هذا البرنامج سيظهر لنا :

```
0 Maître
1 Corbeau
2 sur
3 un
4 arbre
5 perché
```

نتيجة هامة من الطباعة الديناميكية

كما لاحظنا سابقا (في الصفحة 133)، نوع المتغير المستخدم مع العبارة **for** سيتم إعادة تعريفه بإستمرار في الدورات : حتى لو كانت عناصر القائمة بأنواع مختلفة، يمكننا تكرار هذه القائمة بمساعدة **for** بدون أي خطأ، لأن نوع المتغير في الدورات (الحلقات) سوف يتم تعديله تلقائيا إلى نوع العنصر الذي يتم قراءته. على سبيل المثال :

```
>>> divers = [3, 17.25, [5, 'Jean'], 'Linux is not Windoze']
>>> for item in divers:
...     print(item, type(item))
...
3 <class 'int'>
17.25 <class 'float'>
[5, 'Jean'] <class 'list'>
Linux is not Windoze <class 'str'>
```

في المثال أعلاه، استخدمنا الدالة المدمجة **type()** لإظهار أن المتغير **item** يتغير مع كل تكرار (دورة) (و قد أصبح هذا ممكننا من خلال الطباعة الديناميكية للمتغيرات في بيثون).

العمليات على القوائم

يمكننا تطبيق العوامل + (الجمع) و* (الضرب) :

```
>>> fruits = ['orange','citron']
>>> legumes = ['poireau','oignon','tomate']
>>> fruits + legumes
['orange', 'citron', 'poireau', 'oignon', 'tomate']
```

```
>>> fruits * 3
['orange', 'citron', 'orange', 'citron', 'orange', 'citron']
```

العامل * مفيد جدا لإنشاء قائمة من n عناصر متطابقة :

```
>>> sept_zeros = [0]*7
>>> sept_zeros
[0, 0, 0, 0, 0, 0, 0]
```

على سبيل المثال، أنت تريد صنع قائمة **B** الذي تحتوي على نفس العدد من العناصر في قائمة **A**. يمكنك إنجاز هذا بطرق مختلفة، ولكن أبسطها هي : `B = [0]*len(A)`.

اختبار العضوية

يمكنك بسهولة تحديد ما إذا كان العنصر هو جزء من قائمة أو لا بمساعدة العبارة in (هذه العبارة يمكن استخدامها مع المتسلسلات) :

```
>>> v = 'tomate'
>>> if v in legumes:
...     print('OK')
...
OK
```

نسخ لائحة

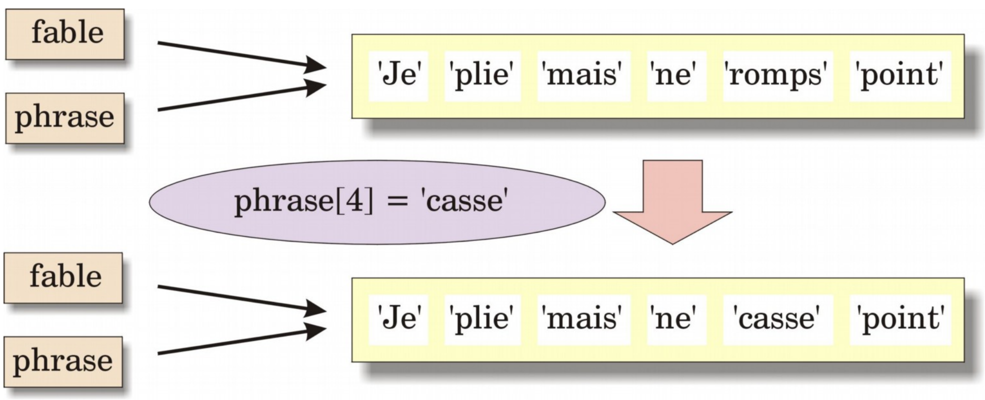
نفترض أن لديك قائمة تريد نسخها في متغير جديد يسمى **phrase**. فإن أول فكرة تتبادر إلى ذهنك هي أن تكتب مهمة بسيطة مثل :

```
>>> phrase = fable
```

من خلال القيام بذلك، اعلم أنك لم تقم بصنع نسخة أصلية. بعد هذه التعليمة، لا توجد سوى قائمة واحدة في ذاكرة الحاسوب. أنت لم تقم سوى بصنع مرجع بسيط لهذه القائمة. حاول على سبيل المثال :

```
>>> fable = ['Je','plie','mais','ne','romps','point']
>>> phrase = fable
>>> fable[4] ='casse'
>>> phrase
['Je', 'plie', 'mais', 'ne', 'casse', 'point']
```

إذا كان المتغير **phrase** يحتوي على نسخة أصلية من القائمة، سوف تكون هذه النسخة مستقلة عن النص الأصلي، وينبغي ألا يتم تعديلها بتعليمة مثل التي بالسطر الثالث، التي تطبق على المتغير **fable**. يمكنك تجربة المزيد من التغييرات الأخرى، سواء على محتويات **fable**، أو على محتويات **phrase**. في جميع الأحوال، سوف تجد أن التعديلات عدلت أيضا على الأخرى، والعكس بالعكس .



في الواقع، **fable** و **phrase** تم تعيين كلاهما في كائن واحد في الذاكرة. لوصف هذه الحالة، يقول علماء الحاسوب أن **phrase** هو اسم مستعار لـ **fable**.

سوف نرى لاحقا، استخدام الاسم المستعار. أما الآن، سوف نتعلم تقنية عمل نسخة أصلية (فعلية) لقائمة. مع المفاهيم المذكورة أعلاه، يجب أن تكون قادرا على إيجاد واحدة بنفسك .

ملاحظة بسيطة حول تركيب الجملة

بيثون يسمح لك "بتوسيع" تعليمة طويلة على عدة أسطر، فإذا استمررت بترميز شيء ما محدد بواسطة زوج من الأقواس، أو المعقوفين، أو قوسين (نصف مربع). يمكنك معالجة العبارات بين قوسين، أو تعريف قوائم طويلة، أو tuples كبيرة أو قواميس كبيرة(انظر أدناه). مستوى مسافة البادئة غير مهم : المفسر يكشف عن نهاية العبارة حيث يتم إغلاق زوج المركب.

هذه الميزة تسمح لك بتحسين إمكانية قراءة برامجك. على سبيل المثال :

```
couleurs = ['noir', 'brun', 'rouge',
            'orange', 'jaune', 'vert',
            'bleu', 'violet', 'gris', 'blanc']
```

تمارين

36.10 انظر في القوائم التالية :

```
t1 = [31,28,31,30,31,30,31,31,30,31,30,31]
t2 = ['Janvier','Février','Mars','Avril','Mai','Juin',
      'Juillet','Août','Septembre','Octobre','Novembre','Décembre']
```

اكتب برنامجا صغيرا يقوم بإدراج جميع عناصر القائمة الأولى في القائمة الثانية ، بحيث يتم فرز كل اسم شهر مع عدد أيامه :

```
['Janvier',31,'Février',28,'Mars',31, etc.].
```

37.10 اصنع قائمة **A** تحتوي على بضعة عناصر. قم بعمل نسخة منها في متغير جديد **B**. اقتراح : قم أولا بصنع القائمة **B** بنفس طول القائمة **A** لكنها لا تحتوي سوى على أصفار. ثم قم باستبدال كل الأصفار بعناصر من القائمة **A**.

38.10 نفس السؤال، لكن اقتراح آخر : قم أولا بصنع القائمة **B** فارغة. ثم قم بملئها بالعناصر القائمة **A** واحدة تلو الأخرى.

39.10 نفس السؤال، لكن اقتراح آخر : لصنع القائمة **B**، قم بقص شريحة من القائمة **A** une tranche incluant tous تحتوي على كل العناصر (بمساعدة المعامل [:]).

40.10 الرقم الأولي هو الرقم الذي يقبل القسمة على نفسه وعلى واحد. اكتب برنامجا يقوم بعرض جميع الأرقام الأولية ما بين 1 و 1000، باستخدام أسلوب غربال إراتونستين :

- أصنع قائمة من 1000 عنصر، قم بتهيئة كل عنصر على القيمة 1.
- إستعرض هذه القائمة بداية من العنصر الثاني : إذا كان العنصر حلل لديه القيمة 1، ضع 0 لجميع العناصر الباقية، والتي هي مؤشرات مضعفات العدد الصحيح للمؤشر الذي حصلت عليه .
- عندما تستعرض جميع القائمة، مؤشرات العناصر التي بقية 1 سيكونون الأرقام الأولية التي نبحث عنها. في الحقيقة : بداية من المؤشر 2، ستلغي جميع العناصر الزوجية : 4، 6، 8، 10 ...إلخ. مع المؤشر 3، سوف تقوم بإلغاء جميع العناصر المؤشر 6، 9، 12، 15 ...إلخ، وهكذ. والعناصر التي بقيت 1 هي أرقام أولية .

الأرقام العشوائية - المدرج الإحصائي

معظم البرامج تقوم بعمل الشيء نفسه في كل مرة تقوم بتشغيلها. وتسمي هذه البرامج بالحتمية (المحددة). الحتمية هي شيء جيد : بالطبع نحن نريد نفس السلسلة من العمليات الحسابية تطبق على نفس المعطيات تؤدي دائما إلى نفس النتيجة. لبعض التطبيقات، إن الحاسوب صعب تكهنه. على سبيل المثال الألعاب هي مثال واضح، هنالك العديد من الأخرين.

على عكس المظاهر، فإنه ليس من السهل كتابة خوارزمية هي حقا غير حتمية (و هذا معناه أن تنتج نتيجة لا يمكن التنبؤ بها). ومع ذلك، هنالك تقنيات رياضية لمحاكات أكثر أو أقل نتيجة الصدفة. لقد كتبت كتب بالكامل لشرح كيفية إنتاج عشوائي "بنوعية جيدة". ونحن بالطبع لن نضع سؤال كهذا.

في وحدة **random**، يقدم لك بيثون مجموعة متنوع من الدالات لتوليد أرقام عشوائية تتبع توزيعات رياضية مختلفة. نحن لن نجرب هنا سوى بعضها. اطلع على وثائق بيثون على الشبكة لمعرفة المزيد. يمكنك استدعاء جميع الدالات في الوحدة بـ :

```
>>> from random import *
```

الدالة `random` لوحدة `random` تسمح لك بإنشاء أعداد حقيقية عشوائية ذات قيمة مابين 0 و 1. البرامتر هو حجم القائمة المطلوبة :

```
>>> def list_aleat(n):
...     s = [0]*n
...     for i in range(n):
...         s[i] =random()
...     return s
...
>>> list_aleat(3)
[0.37584811062278767, 0.03459750519478866, 0.714564337038124]
>>> list_aleat(3)
[0.8151025790264931, 0.3772866844634689, 0.8207328556071652]
```

يمكنك أن ترى أننا أولا أخذنا جزءا ببناء قائمة من الأصفار بطول n ، ثم قمنا باستبدال الأصفار بأرقام عشوائية .

تمارين

41.10 أعد كتابة الدالة **list_aleat()** في الأعلى، باستخدام الأسلوب **append()** لصنع قائمة جزءا جزءا بداية من قائمة فارغة (بدلا من استبدال الأصفار من قائمة موجودة سابقا كما فعلنا قبل قليل).

42.10 اكتب الدالة **imprime_liste()** التي تسمح بإظهار جميع العناصر الموجودة في قائمة بأي حجم سطرا سطرا. اسم القائمة سيكون في البرامتر ، استخدم هذه الدالة لطباعة قائمة من الأرقام العشوائية التي تم صنعها بواسطة الدالة **list_aleat()**. على سبيل المثال التعليمة `imprime_liste(liste_aleat(8))` سوف تقوم بعرض عمود من 8 أرقام حقيقية عشوائية .

هل الأرقام التي تم تولدها هي فعلا عشوائية ؟ هذا الشيء صعب قوله. نحن لم نفعل سوى لعدد قليل من القيم، لا يمكننا التحقق من هذا. ومن جانب آخر، إذا استخدمنا الدالة **random()** مرات عديدة، نتوقع أن يكون نصف القيم المنتجة هي أكبر من 0.5 (و النصف الأخر أقل).

نركز على هذا المنطق. القيم التي يتم الحصول عليها هي دائما في نطاق 0 - 1. مشاركة هذا الفاصل الزمني في 4 أجزاء متساوية : من 0 إلى 0.25 و من 0.25 إلى 0.5 ومن 0.5 إلى 0.75 ومن 0.75 إلى 1 فإذا وضعنا عددا كبيرا من القيم العشوائية، نحن نتوقع أنه سيكون هنالك الكثير من الانخفاض في الكسور الأربعة. ويمكننا تعميم هذا المنطق إلى أي رقم أي كسر، طالما أنهم متساوون .

تمرين

43.10 اكتب برنامجا يتحقق من عمل مولد الأرقام العشوائية في بيثون باستخدام النظرية المذكورة أعلاه . البرنامج سيقوم بالتالي :

- أطلب من المستخدم عدد القيم العشوائية التي سيتم إنشاؤها بمساعدة الدالة **random()**. وسيكون من المثير للاهتمام أن يوفر البرنامج عددا افتراضيا (1000 على سبيل المثال) .

- اطلب من المستخدم كم يريد لامشاركة في مجموعة قيم الكسور الممكنة (و هذا معناه ما بين 0-1). هنا أيضا، يجب أن تقدم عدد الكسور الافتراضي (5 على سبيل المثال). يمكنك أن تحدد للمستخدم ما بين 2 و 10\1 عدد القيم العشوائية.
- قم ببناء قائمة من N عدادات (N ستكون عدد من الكسور المطلوبة). وسيتم تهيئة كل واحدة منهم إلى الصفر.
- إسحب عشوائيا جميع القيم المطلوبة، بمساعدة الدالة **random()** ، وقم بتخزين هذه القيم داخل قائمة .
- قم بتدوير قائمة القيم التي تم سحبها عشوائيا (حلقة)، وقم بإجراء اختبار لكل واحد منها لتحديد ما هي جزء من فترة الفاصلة 0-1 هي عليها. يزداد العداد واحد واحد .
- عند الانتهاء، اعرض حالة كل عداد .

مثال على نتائج التي يتم عرضها من برنامج من هذا النوع :

```
Nombre de valeurs à tirer au hasard (défaut = 1000) : 100
Nombre de fractions dans l'intervalle 0-1 (entre 2 et 10, défaut =5) : 5
Tirage au sort des 100 valeurs ...
Comptage des valeurs dans chacune des 5 fractions ...
11 30 25 14 20
Nombre de valeurs à tirer au hasard (défaut = 1000) : 10000
Nombre de fractions dans l'intervalle 0-1 (entre 2 et 1000, défaut =5) : 5
Tirage au sort des 10000 valeurs ...
Comptage des valeurs dans chacune des 5 fractions ...
1970 1972 2061 1935 2062
```

أسلوب جيد لهذه المشكلة من خلال تصور دالات بسيطة لكتابتها لحل جزء أو آخر من المشكلة، ثم استخدامها لأشياء أكبر.

على سبيل المثال، تستطيع أولا أن تحاول تعريف الدالة **numeroFraction()** التي تحدد أي جزء ما بين 0-1 (قيمةالمستمدة). هذه الدالة تأخذ برامترين (القيمة المستمدة، عدد الكسور التي يتم اختيارها من قبل المستخدم) ويقوم بإرجاع مؤشر العداد لزيادته (هذا معناه رقم الكسر). قد يكون هنالك منطق رياضي بسيط الذي يسمح لك بتحديد المؤشر الكسر من هذين البرامتران. بما في ذلك الدالة المدمجة **int()**ـ التي تسمح لك بتحويل عدد حقيقي إلى عدد صحيح بإزالة الجزء العشري.

فإذا لم تجدها، فكرة أخرى مثير للاهتمام، إبدأ ببناء قائمة تحتوي على القيم "المحاور" التي تحدد الكسور المحددة (على سبيل المثال 0 – 0,25 – 0,5 – 0,75 - 1 في حالة 4 كسور). معرفة هذه القيم قد يسهل كتابة الدالة **numeroFraction()** التي نريد أن نطورها.

إذا كان لديك المزيد من الوقت، يمكنك أيضا صنع نسخة رسومية من البرنامج، والتي سوف تعرض لك النتائج على رسم بياني (الرسم البياني "بالعصا")

سحب الأعداد الصحيحة عشوائيًا

عندما تطور مشاريعك الشخصية، سوف تحتاج إلى الكثير من الأحيان إلى دالة تسمح لك عشوائيا بسحب عدد صحيح في حدود معينة. على سبيل المثال، فإذا أردت كتابة برنامج للعبة أوراق اللعب التي يتم سحبها عشوائيا (اللعبة العادية 52 ورقة)، سوف تستخدم بكل تأكيد دالة يمكنها سحب عدد صحيح عشوائيا ما بين 1 و 52 .

يمكنك القيام بهذا عن طريق الدالة **()randrange** للوحدة **random**. هذه الدالة تستخدم مع 1 أو 2 أو 3 برامترات.

باستخدام برامتر واحد، تقوم بإرجاع عدد صحيح ما بين 0 والقيمة البرامتر ناقص 1. على سبيل المثال، **(randrange(6** سوف تقوم بإرجاع عدد ما بين 0 و 5.

باستخدام برامترين، العدد الذي سيتم إرجاعه سيكون ما بين البرامتر الأول والبرامتر الثاني ناقص واحد. على سبيل المثال، **(randrange(2, 8** تقوم بإرجاع عدد ما بين 2 و 7.

و إذا أضفنا برامتر ثالث، فإنه يشير إلى أن العدد الذي يجب سحبه يجب أن يكون من مجموعة من الأعداد محددة من الأعداد الصحيحة، ويتم فصلها عن بعض بفاصل معين، الذ تم تعريفه بالبرامتر الثالث. على سبيل المثال، **(randrange(3, 13, 3** سوف تقوم بإرجاع عدد من 3، 6، 9، 12 :

```
>>> from random import randrange
>>> for i in range(15):
...     print(randrange(3, 13, 3), end =' ')
...
12 6 12 3 3 12 12 12 9 3 9 3 9 3 12
```

تمارين

44.10 اكتب سكريبت يقوم بسحب عشوائيا أوراق اللعب. اسم الورقة التي يتم سحبها يجب أن يكون قد عرض بشكل صحيح، "بوضوح". سيقوم البرنامج بعرض على سبيل المثال :

```
Frappez <Enter> pour tirer une carte :
Dix de Trèfle
Frappez <Enter> pour tirer une carte :
As de Carreau
Frappez <Enter> pour tirer une carte :
Huit de Pique
Frappez <Enter> pour tirer une carte :
```

إلخ.

المصفوفات المغلقة (tuples)

لقد درسنا حتى الآن نوعين من المعطيات المركبة : السلاسل، والتي هي مركبات حروف، والقوائم، والتي هي مركبات عناصر من أي نوع. ويبج أن تتذكر فرق آخر ما بين السلاسل والقوائم : غير ممكن تغيير الأحرف في سلسلة، إذا يمكنك تعديل(تحرير)

العناصر في سلسلة. وبعبارة أخرة، القوائم هي متسلسلات قابلة للتعديل، والسلاسل النصية هي متسلسلان غير قابلة للتعديل، على سبيل المثال :

```
>>> liste =['jambon','fromage','miel','confiture','chocolat']
>>> liste[1:3] =['salade']
>>> print(liste)
['jambon', 'salade', 'confiture', 'chocolat']

>>> chaine ='Roméo préfère Juliette'
>>> chaine[14:] ='Brigitte'

         ***** ==> Erreur: object doesn't support slice assignment  *****
```

لقد حاولنا تغيير نهاية السلسلة النصية، لكننا لم ننجح. السبيل الوحيد لتحقيق أهدافنا هو بصنع سلسلة جديدة، ونقوم بنسخ ما نريد تغييره :

```
>>> chaine = chaine[:14] +'Brigitte'
>>> print(chaine)
Roméo préfère Brigitte
```

توفر لك بيثون نوعا من البيانات يدعى المصفوفة المغلقة ***tuple***[61]، وهو يشبه كثيرا القوائم، لكنه مثل السلاسل، لا يمكن تعديله.

من المنظور اللغوي، المصفوفة المغلقة (tuple) هي مجموعة من العناصر مفصولة بفواصل :

```
>>> tup = 'a', 'b', 'c', 'd', 'e'
>>> print(tup)
('a', 'b', 'c', 'd', 'e')
```

على الرغم من أنه غير ضروري، لكن من المستحسن تحديد المصفوفة المغلقة (tuple) بزوج من الأقواس، مثل الدالة **print()** في بيثون. هذا ببساطة لزيادة إمكانية قراءة الكود، لكن هذا مهم .

```
>>> tup = ('a', 'b', 'c', 'd', 'e')
```

العمليات على المصفوفات المغلقة (tuples)

لا يمكن أن تعمل العمليات على المصفوفة المغلقة (tuple) بناء جملة المصفوفات المغلقة (tuples) ليس مماثل للقوائم، لأن المصفوفات المغلقة (tuples) غير قابلة للتغيير :

```
>>> print(tup[2:4])
('c', 'd')
>>> tup[1:3] = ('x', 'y')          ==> ***** خطأ! *****
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> tup = ('André',) + tup[1:]
```

---

61 هذا المصطلح ليس كلمة إنكليزية عادية ؛ هو لفظ حاسوبي جديد .

```
>>> print(tup)
('André', 'b', 'c', 'd', 'e')
```

لاحظ أنه يجب عليك دائما على الأقل فاصلة واحدة لتعريف المصفوفة المغلقة (tuple) (المثال الأخير فوق يستخدم مصفوفة مغلقة (tule)يحتوي على عنصر واحد: 'André').

يمكنك تحديد طول المصفوفة المغلقة (tuple) بمساعدة **len()**، التدوير بمساعدة الحلقة **for**، استخدم التعليمة **in** لمعرفة ما إذا كان العنصر المقدم هو جزء، إلخ ...، تماما مثلما كنت تفعل مع القائمة. عمليات الجمع والضرب تعمل إذا ؟ لكن لأن المصفوفات المغلقة (tuples) غير قابلة للتغيير (التحرير)، لا يمكن استخدام التعليمة **del** ولا الأسلوب **remove()**مع المصفوفات المغلقة (tuples) :

```
>>> tu1, tu2 = ("a","b"), ("c","d","e")
>>> tu3 = tu1*4 + tu2
>>> tu3
('a', 'b', 'a', 'b', 'a', 'b', 'a', 'b', 'c', 'd', 'e')
>>> for e in tu3:
...     print(e, end=":")
...
a:b:a:b:a:b:a:b:c:d:e:
>>> del tu3[2]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object doesn't support item deletion
```

سوف تفهم فائدة استخدام المصفوفات المغلقة (tuples) تدريجيا. نلاحظ ببساطة أننا نفضله عن قوائم لأن البيانات التي يتم تمريرها لن تتم تعديلها بالخطأ في البرنامج. وبالإضافة إلى ذلك، المصفوفات المغلقة (tuples) أقل "جشع" على موارد النظام (أي أنه تأخذ مساحة أقل في ذاكرة، ويمكن معالجتها بسرعة من قبل المفسر) .

القواميس

أنواع البيانات المركبة التي ذكرناها حتى الآن هي : السلاسل والقوائم والمصفوفات المغلقة (tuples) وهم جميعا من المتسلسلات، وهذا معناه تسلسل مرتب من العناصر. في التسلسل، من السهل الوصول إلى أي عنصر من خلال مؤشره (عدد صحيح)، لكن شرط معرفة موقعها.

القواميس التي نكتشفها الآن هي نوع آخر من المركبات. وهي تبدو مثل القوائم إلى حد كبير (يمكن تعديلها مثل القوائم)، لكنها ليست من المتسلسلات. العناصر التي نقوم بحفظها لن تكون في ترتيب ثابت. ومن جهة أخرى، يمكننا الوصول إلى أي وحدة منها (عنصر)بمساعدة مؤشر خاص يسمى المفتاح، والذي قد يكون من الحروف أو الأرقام أو حتى مركب بشروط معينة.

كما هو الحال في القائمة. العناصر التي يتم تخزينها في القاموس يمكن أن تكون من أي نوع. هذا يعني أنها يمكن أن تكون من القيم الرقمية أو سلاسل أو قوائم أو أنفاق (tuples) أو قواميس أو حتى دالات أو أصناف أو المثيل (سوف نراه لاحقا)[62].

---

[62] القوائم والمصفوفات المغلقة قد تحتوي أيضا على قواميس، ودالات، وأصناف ومثيلات، ونحن لن نذكرها حتى الآن، حتى لا تقيد العرض التقديمي .

إنشاء قاموس

على سبيل المثال، سوف ننشئ قاموس للغة، لترجمة المصطلحات الحاسوبية من اللغة الإنكليزية إلى اللغة الفرنسية.

و بما أن القواميس قابلة للتعديل، يمكننا البدء بإنشاء قاموس فارغ، ثم نقوم بملئه تدريجيا. من منظور لغوي (تركيب الجملة)، اعلم أن القاموس يتم وضع عناصره في زوج من الأقواس (معقوف). وللإشارة إلى القاموس الفارغ بـ { } :

```
>>> dico = {}
>>> dico['computer'] = 'ordinateur'
>>> dico['mouse'] ='souris'
>>> dico['keyboard'] ='clavier'

>>> print(dico)
{'computer': 'ordinateur', 'keyboard': 'clavier', 'mouse': 'souris'}
```

و كما ترى في السطر الأخير أعلاه، يظهر القاموس في بيثون على شكل سلسلة من العناصر مفصولة بفواصل، يتم إحاطة كل هذا بقوسين (معقوفين). كل عنصر من هذه العناصر هو في حد ذاته يشكل زوج من الكائنات : المؤشر والقيمة مفصولة بنقطتين.

في القاموس، يسمى المؤشر بالمفتاح، والعناصر تدعى إذا بزوج من مفتاح القيمة. القاموس في مثالنا، المفاتيح والقيم سلاسل نصية.

يرجى ملاحظة أن ترتيب العناصر الموجود في السطر الأخير لا طابق الترتيب الذي قدمناه .

```
>>> print(dico['mouse'])
souris
```

يرجى ملاحظة أن ترتيب أي عنصر لا يطابق الذي قدمناه سابقا. وهذه ليست لها أي أهمية : وهذا معناه أننا لن نستخرج قيمة أحد عناصر القاموس بمساعدة رقم ترتيبه، بدلا من ذلك نستخدم المفاتيح :

نلاحظة أيضا، على عكس القوائم، فإنه ليس من الضروري استدعاء أسلوب معين (مثل **append()**) لإضافة عنصر جديد إلى القاموس : فقط قم بصنع زوج جديد من مفتاح-قيمة .

العمليات على القواميس

أنت تعرف كيفية إضافة عنصر إلى قاموس. لإزالة عنصر، نستخدم التعليمة المدمجة **del**. اصنع على سبيل المثال قاموسا آخر، في هذه المرة يحتوي على مخزون من الفاكهة المؤشرات (أو المفاتيح) سيكونون أسماء فواكهة، والقيم ستكون كتل هذه الفواكهة المدرجة في المخزون (القيم ستكون هذه المرة قيم رقمية) .

```
>>> invent = {'pommes': 430, 'bananes': 312, 'oranges' : 274, 'poires' : 137}
>>> print(invent)
{'oranges': 274, 'pommes': 430, 'bananes': 312, 'poires': 137}
```

فإذا أراد سيدك تصفية جميع التفاح ولا بيعها، يمكننا إزالة من القاموس :

```
>>> del invent['pommes']
>>> print(invent)
{'oranges': 274, 'bananes': 312, 'poires': 137}
```

الدالة المدمجة **()len** متكاملة مع القواميس : تقوم بإرجاع عدد العناصر:

```
>>> print(len(invent))
3
```

اختبار الانتماء

مثل ما يحدث للسلاسل والقوائم والمصفوفات المغلقة (tuples)، التعليمة **in** تعمل مع القواميس. فهي تسمح لم إذا كان القاموس يحتوي على مفتاح محدد[63]:

```
>>> if "pommes" in invent:
...     print("Nous avons des pommes")
... else:
...     print("Pas de pommes. Sorry")
...
Pas de pommes. Sorry
```

القواميس هي كائنات

يمكننا أن نطبق على القوائم العديد من الأساليب الخاصة :

الأسلوب **()keys** يقوم بإرجاع تسلسل المفاتيح المستخدمة في القاموس. هذا التسلسل يمكن استخدامه في العبارات أو تحويلها إلى قائمة أو إلى مصفوفات مغلقة إذا لزم الأمر، عن طريق دالات المدمجة مثل **()list** أو **()tuple** :

```
>>> print(dico.keys())
dict_keys(['computer', 'mouse', 'keyboard'])
>>> for k in dico.keys():
...     print("clé :", k, " --- valeur :", dico[k])
...
clé : computer  --- valeur : ordinateur
clé : mouse  --- valeur : souris
clé : keyboard  --- valeur : clavier
>>> list(dico.keys())
['computer', 'mouse', 'keyboard']
>>> tuple(dico.keys())
('computer', 'mouse', 'keyboard')
```

و بشكل مماثل، فإن الأسلوب **()values** يقوم بإرجاع سلسلة من القيم المخزنة في القواميس :

```
>>> print(invent.values())
dict_values([274, 312, 137])
```

أما بالنسبة للأسلوب **()items**، فهو يقوم باستخراج من القاموس سلسلة معادلة للمصفوفات المغلقة (tuples). وهذا الأسلوب سيكون مفيدا جدا فيما بعد، عندما نريد تكرار قاموس بمساعدة حلقة :

```
>>> invent.items()
dict_items([('poires', 137), ('bananes', 312), ('oranges', 274)])
```

63 في الإصدارات السابقة لبيثون، كان من الضروري استدعاء أسلوب معين (الأسلوب **()has_key**) لأداء هذا الاختبار.

```
>>> tuple(invent.items())
(('poires', 137), ('bananes', 312), ('oranges', 274))
```

الأسلوب **copy()** يسمح لك بصنع نسخة طبق الأصل من قاموس. يجب أن تعرف إذا قدمت متغيرا جديدا وربطه بالقاموس سيكون هذا فقط مرجعا للكائن نفسه، وليس كائنا جديدا. سبق وأن تحدثنا على هذا الأمر في القوائم (انظر إلى الصفحة Error: Reference source not found). على سبيل المثال، العبارة بالأسفل لا تعرف قاموسا جديدا (عكس المظاهر) :

```
>>> stock = invent
>>> stock
{'oranges': 274, 'bananes': 312, 'poires': 137}
```

فإذا قمنا بتغيير **invent**، سيتم تغيير **stock** أيضا، والعكس بالعكس (هذان الاسمان يشيران إلى نفس كائن القاموس في ذاكرة الحاسوب) :

```
>>> del invent['bananes']
>>> stock
{'oranges': 274, 'poires': 137}
```

لصنع نسخة حقيقية (مستقلة) لقاموس موجود مسبقا، يجب عليك استخدام الأسلوب **copy()** :

```
>>> magasin = stock.copy()
>>> magasin['prunes'] = 561
>>> magasin
{'oranges': 274, 'prunes': 561, 'poires': 137}
>>> stock
{'oranges': 274, 'poires': 137}
>>> invent
{'oranges': 274, 'poires': 137}
```

تدوير قاموس

يمكنك استخدام حلقة التكرار **for** لمعالجة جميع عناصر في قاموس، لكن انتبه :

- خلال التكرار، المفاتيح المستخدمة في القاموس هي التي سيتم تعيينها تبعا لمتغير العمل، وليس القيم.
- الترتيب الذي سيتم استخراج العناصر به لا يمكن التنبؤ به (لأن القاموس ليس متسلسل) .

مثال:

```
>>> invent ={"oranges":274, "poires":137, "bananes":312}
>>> for clef in invent:
...     print(clef)
...
poires
bananes
oranges
```

إذا كنت ترغب في أن تقوم بمعالجة على القيم، يكفي أن تقوم باسترداد كل واحد من المفاتيح المطابقة :

```
>>> for clef in invent:
...     print(clef, invent[clef])
```

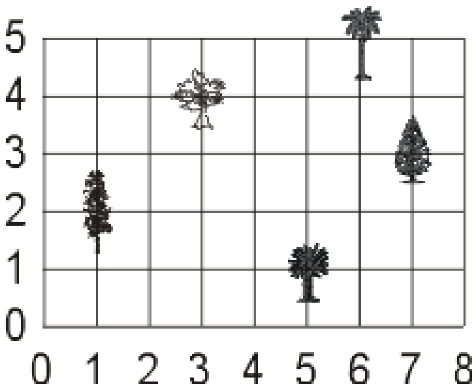
```
...
poires 137
bananes 312
oranges 274
```

هذا النهج ليس مثاليا، سواء من حيث الأداء أو حتى من وجهة نظر الوضوح. فمن المستحسن بدلا من ذلك استخدام طريقة **items()** الذي تم وصفه في القسم السابق :

```
for clef, valeur in invent.items():
    print(clef, valeur)
...
poires 137
bananes 312
oranges 274
```

في هذا المثال، الأسلوب **items()** يتم تطبيقه على قاموس **invent** الذي يقوم بإرجاع سلسلة من المصفوفات المغلقة (tuples) (المفتاح، القيمة). يتم تنفيذ هذه الدورة على هذه القائمة بمساعدة حلقة التي تقوم بفحص كل عنصر من عناصر المصفوفة المغلقة (tuples) .

عناصر المصفوفة المغلقة (tuples) .



حتى الآن وصفنا أن مفاتيح القواميس هي من نوع سلسلة (string). في الحقيقة يمكننا استخدام كمفتاح من أي نوع من البيانات الغير قابلة للتغيير : أعداد صحيحة، أعداد حقيقية، سلاسل نصية، وحتى المصفوفات المغلقة (tuples).

على سبيل المثال، نريد أن نتعرف على الأشجار الملحوظة التي توجد في حقل مستطيل كبير. لهذا يمكننا استخدام قاموس، ومفاتيحه ستكون مصفوفات مغلقة (tuples) التي تشير إلى إحداثيات **x,y** لكل شجرة:

```
>>> arb = {}
>>> arb[(1,2)] = 'Peuplier'
>>> arb[(3,4)] = 'Platane'
>>> arb[6,5] = 'Palmier'
>>> arb[5,1] = 'Cycas'
>>> arb[7,3] = 'Sapin'

>>> print(arb)
{(3, 4): 'Platane', (6, 5): 'Palmier', (5, 1): 'Cycas', (1, 2): 'Peuplier',
 (7, 3): 'Sapin'}

>>> print(arb[(6,5)])
palmier
```

قد تلاحظ أننا قللنا من الكتابة بداية من السطر الثالث, بالاستفادة من أقواس المصفوفات المغلقة (tuples) الاختيارية (استخدمها بحذر !)

في هذا النوع من البناء، نأخذ في اعتبارنا أن القاموس يحتوي فقط على بعض العناصر من أزواج الإحداثيات. وعلاوة على ذلك، لا يوجد شيء. وبالتالي، إن كنا نريد الاستعلام في القاموس لمعرفة مكان غير موجود، على سبيل المثال الإحداثيات (2,1)، فإننا نحصل على خطأ :

```
>>> print(arb[1,2])
Peuplier
>>> print(arb[2,1])

          ***** Erreur : KeyError: (2, 1)  *****
```

لحل هذا المشكلة الصغيرة، يمكننا استخدام الأسلوب **get()** :

```
>>> arb.get((1,2), 'néant')
Peuplier
>>> arb.get((2,1), 'néant')
néant
```

البرامتر الأول الذي يتم تمريره هو لهذا الأسلوب هو مفتاح البحث، أما البرامتر الثاني فهي القيمة التي نريد الحصول عليها إذا كان المفتاح غير موجود في القاموس .

القواميس ليست متسلسلة

كما رأيتم أعلاه، لا يتم ترتيب عناصر القاموس في أي ترتيب معين. العمليات مثل التسلسل والاستخراج (من مجموعة من العناصر المتصلة) لا يمكن تطبيقها هنا ببساطة. فإذا حاولت في أي حال من الأحوال، سيقوم بيثون بطباعة خطأ عند تشغيل كود :

```
>>> print(arb[1:3])

         ***** Erreur : TypeError: unhashable type *****
```

و رأيتم أيضا أنه يكفي لتعيين مؤشر جديد (مفتاح جديد) لإضافة مدخلات إلى القاموس. وهذه لا تعمل مع القوائم[64] :

```
>>> invent['cerises'] = 987
>>> print(invent)
{'oranges': 274, 'cerises': 987, 'poires': 137}

>>> liste =['jambon', 'salade', 'confiture', 'chocolat']
>>> liste[4] ='salami'

         ***** IndexError: list assignment index out of range  *****
```

64تذكير : الأساليب التي تسمح بإضافة عناصر إلى القائمة تم شرحها في صفحة 154.

هي ليست من المتسلسلات، ثبت أن القواميس إذا قيم خاصة تم صنعها يتجميع معطيات والتي يتعين على المرء الإضافة عليها أو الحذف منها، في أي ترتيب كانت. وهي تحل محل القوائم عندما يتعلق الأمر بالتعامل مع مجموعات من المعطيات الرقمية، والتي هم غير متسلسلين.

مثال :

```
>>> client = {}
>>> client[4317] = "Dupond"
>>> client[256] = "Durand"
>>> client[782] = "Schmidt"
```

إلخ.

تمارين

45.10 اكتب سكريبت يقوم بصنع نظام قاعدة بيانات صغير يعمل بمساعدة القاموس، هذا النظام يقوم بحفظ أسماء عدد من أصدقائك وأعمارهم وطولهم. السكريبت الخاص بك يجب عليه أن يحتوي على دالتين : الأولى لملء القاموس، والثانية للاطلاع. في دالة الملء، استخدم حلقة لقبول المعطيات من المستخدم.

في القاموس، اسم الطالب سيكون مفتاح الوصول، والقيم ستكون أنفاق (tuples) (العمر، الطول)، حيث يتم التعمير على العمر بالسنوات (عدد صحيح)، والطول بالأمتار (عدد حقيقي). دالة الإستشارة تشمل أيضا حلقة، حيث يستطيع المستخدم إدخال أي اسم ليتم إرجاع له زوجين (العمر، الطول) المقابل. وستكون نتيجة الاستعلام سطر منسق جيدا، على سبيل المثال (الاسم: جين دوت - العمر: 15 سنة - الطول: 1.74 متر). لتحقيق ذلك، استخدم تنسيق السلاسل النصية التي شرحناها في الصفحة 149.

46.10 اكتب دالة تبدل مفاتيح بقيم القاموس (تسمح لك على سبيل المثال بتحويل قاموس إنكليزي\فرنسي بقاموس فرنسي\إنكليزي). نفترض أن القاموس لا يحتوي على قيم متطابقة عديدة .

القواميس هي أداة لإنشاء مدرج بياني أنيق .

على سبيل المثال، لنفترض أننا نريد إنشاء رسم بياني يمثل تكرار كل حرف من الأبجدية في نص المقدم. خوارزمية التي تقوم بهذا العمل هي بسيطة للغاية إذا كنت تريد بنائها بالاعتماد على قاموس :

```
>>> texte ="les saucisses et saucissons secs sont dans le saloir"
>>> lettres ={}
>>> for c in texte:
...     lettres[c] =lettres.get(c, 0) + 1
...
>>> print(lettres)
{'t': 2, 'u': 2, 'r': 1, 's': 14, 'n': 3, 'o': 3, 'l': 3, 'i': 3, 'd': 1, 'e': 5, 'c':
3, ' ': 8, 'a': 4}
```

نبدأ من خلال إنشاء قاموس فارغ: **lettres**. ثم سنستخدم لملء هذا القاموس الأبجدية كمفتاح. القيم التي نريد تخزينها لك مفاتيح ستكون تكرار الأحرف في النص المقابل. لحساب هذا، سوف نقوم بتدوير سلسلة النصية **texte**. لكل حرف، سوف نطلب من القاموس بمساعدة الأسلوب **get()**، باستخدام الحرف كمفتاح لقراءة التكرار الموجود لهذا الحرف. إذا كانت القيمة غير موجودة سوف يعيد لنا الأسلوب **get()** قيمة فارغة. في جميع الحالات، سنقوم بزيادة القيمة الموجود، ونخزنها في القاموس، في المكان الذي يتوافق مع المفتاح (هذا معناه إلى الحرف الذي تتم معالجته).

لتحسين عملنا، نستطيع أن نعرض الرسم البياني في ترتيب أبجدي. للقيام بذلك، سوف نفكر على الفور باستخدام أسلوب الفرز **sort()**. لكن هذا لا يمكن تطبيقه إلا على القوائم. هذا لا يهم! لقد رأينا في الأعلى كيف يمكننا تحويل القاموس إلى قائمة مصفوفات مغلقة (tuples):

```
>>> lettres_triees = list(lettres.items())
>>> lettres_triees.sort()
>>> print(lettres_triees)
[(' ', 8), ('a', 4), ('c', 3), ('d', 1), ('e', 5), ('i', 3), ('l', 3), ('n', 3), ('o',
3), ('r', 1), ('s', 14), ('t', 2), ('u', 2)]
```

تمارين

47.10 لديك تحت تصرفك أي ملف نصي (ليس كبير جدا). اكتب سكريبت يحسب تكرار كل حرف من الأبجدية في هذا النص (لتسهيل المشكلة تجاهل الحروف المعلمة) .

48.10 قم بتعديل السكريبت بالأعلى بحيث ينشئ جدولا تواجد كل كلمة في النص. نصيحة : في أي نص، لا تفصل الكلمات فقط بالمسافات، لكن حتى بأدوات التنقيط المختلفة. لتبسيط المشكلة، يمكنك البدء باستبدال جميع الرموز بمسافات، تحويل السلسلة الناتجة في قائمة من الكلمات بمساعدة الأسلوب **split()**.

49.10 لديك تحت تصرفك أي ملف نصي (ليس كبيرا جدا). اكتب سكريبت يحلل هذا النص، ويقوم تخزين في قاموس المكان الدقيق لكل كلمة (قم بعد لبأحرف في البداية). إذا كانت كلمة موجودة في أماكن مختلفة، جميع الأماكن يجب تخزينها : كل قيمة في قاموس يجب أن تكون قائمة الأماكن .

التحكم في تدفق التنفيذ باستخدام قاموس

كثيرا ما يحدث أننا نقوم بتنفيذ البرامج في اتجاهات مختلفة، اعتمادا على قيمة المتغير. يمكنك بالطبع التعامل مع هذه المشكلة باستخدام سلسلة من العبارات **if - elif - else**، لكنها يمكن أن تصبح مرهقة جدا وغير أنيقة إذا كنت تتعامل مع عدد كبير من الاحتمالات، على سبيل المثال:

```
materiau = input("Choisissez le matériau : ")

if materiau == 'fer':
    fonctionA()
```

```
elif materiau == 'bois':
    fonctionC()
elif materiau == 'cuivre':
    fonctionB()
elif materiau == 'pierre':
    fonctionD()
elif   ... etc ...
```

لغات البرمجة توفر لك عبارات محددة للتعامل مع هذه المشكلة، مثل العبارات switch أو case في السي أو في باسكال. بيثون لا توفر لك أي واحد، لكن يمكنك الحصول عليها بمساعدة قائمة (لقد أعطينا شرح مفصل في الصفحة Error: Reference source not found)، أو أفضل من ذلك عن طريق قاموس. على سبيل المثال :

```
materiau = input("Choisissez le matériau : ")

dico = {'fer':fonctionA,
        'bois':fonctionC,
        'cuivre':fonctionB,
        'pierre':fonctionD,
           ... etc ...}

dico[materiau]()
```

التعليمتان في الأعلى يمكن جمعها في تعليمة واحد فقط، لكننا جعلناها منفصلة لتفصيل الآلية :

- العبارة الأولى تعرف قاموس **dico** حيث أن المفاتيح هي الاحتمالات المختلفة للمتغير **materiau**، والقيم هي أسماء الدالات التي يجب استدعاؤها. لاحظ أنها سوى أسماء الدالات، ومن المهم أن لا تضع القوسين في هذه الحالة (و إلا سوف يقوم بيثون بتنفيذ جميع الدالات في وقت صنع القاموس) .
- العبارة الثانية تقوم باستدعاء الدالة المقابلة للاختيار بمساعدة الدالة **materiau**. اسم الدالة سيتم استخراجه من القاموس بمساعدة المفتاح، ثم يتم ربط بزوج من الأقواس. بيثون ستعرف إذا أنها استدعاء دالة بشكل تقليدي، ثم يتم تشغيله .

يمكنك تعزيز التقنية في الأعلى باستبدال هذه التعليمة مع مثيلها في الأسفل، الذي يقوم باستدعاء الأسلوب **get()** حالة إذا كان المفتاح المطلوب غير موجود في القاموس(و بهذه الطريقة يمكنك الحصول على ما يعادل تعليمة **else** في نهاية السلسلة الطويلة من **elif**) :

```
dico.get(materiau, fonctAutre)()
```

عندما تكون قيمة المتغير **materiau** لا تتطابق مع أي مفتاح في القاموس، يتم استدعاء الدالة () **fonctAutre()** .

تمارين

50.10 أكمل التمرين 10.46 (نظام قاعدة بيانات صغير) بإضافة دالتين : واحدة لحفظ القاموس الناتج في ملف نصي والثانية لإعادة بناء هذا القاموس من خلال هذا الملف النصي.

كل سطر من الملف النصي يتوافق مع عنصر من القاموس. سيتم تنسيق ذلك بطريقة مفصول جيدا :

المفتاح والقيمة (هذا معناه اسم الشخص جزء، ومجموعة : العمر + الطول، في جزء آخر).

في مجموعة "العمر + الطول"، هذان الاثنان معطيات رقمية. لذا يجب عليك استخدام رمز للفصل، على سبيل المثال "@" للفصل بين المفتاح والقيمة، و"#" للفصل بين معطيات هذه القيمة :

```
Juliette@18#1.67
Jean-Pierre@17#1.78
Delphine@19#1.71
Anne-Marie@17#1.63
```
etc.

51.10 حسّن سكريبت التمرين السابق، باستخدام قاموس بتوجيه تدفق تنفيذ البرنامج في مستوى القائمة الرئيسية. سوف يعرض البرنامج على سبيل المثال :

```
Choisissez :
(R)écupérer un dictionnaire préexistant sauvegardé dans un fichier
(A)jouter des données au dictionnaire courant
(C)onsulter le dictionnaire courant
(S)auvegarder le dictionnaire courant dans un fichier
(T)erminer :
```

اعتمادا على اختيار المستخدم، يتم إذًا استدعاء الدالة المقابلة عن طريق اختيار في قاموس الدالات .

11

الأصناف، الكائنات، الصفات

في الفصول السابقة، لقد التقيت بالفعل عدة مرات مع مفهوم الكائن. وأنت تعرف أن الكائن هو وحدة تم إنشاؤها من صنف مثيل (و هذا معناه هو نوع من "فئة" أو "نوع" كائن). على سبيل المثال، يمكننا أن نجد في مكتبة *Tkinter*، صنف () ***Button*** من خلالها يمكننا صنع أي عدد من الأزرار في النافذة.

سوف ندرس الآن كيف يمكنك تعريف أصناف جديدة من الكائنات. هذا الموضوع صعب نسبيا، ولكننا سوف نقترب تدريجيا، بداية من تعريف أصناف كائنات بسيطة جدا، التي سوف نصقلها.

مثل كائنات الحياة اليومية (الحقيقية)، الكائنات الحاسوبية يمكن أن تكون بسيطة جدا أو معقدة جدا. قد تكون مكونة من أجزاء مختلفة، والتي هي في حد ذاتها كائنات، وهذه جعلت بدورها كائنات أخرى أكثر بساطة، إلخ ...

فائدة الأصناف

الأصناف هي الأدوات الرئيسية للبرمجة الشيئية (Object Oriented Programming أو OOP). هذا النوع من البرمجة يسمح لك بهيكلة البرامج المعقدة عن طريق تنظيمها على أنها مجموعات من كائنات تتفاعل مع بعضها ومع العالم الخارجي.

الفائدة الأولى من هذا النهج من البرمجة يكمن في بناء كائنات مختلفة تستخدم بشكل مستقل عن بعضها البعض (على سبيل المثال من قبل مبرمجين مختلفين) بدون خطر تداخلها. هذه النتيجة تحقق من خلال مفهوم التغليف : الوظيفة الداخلية للكائن والمتغيرات التي تستخدم للقيام بعملها، نوعا ما بشكل مغلق في الكائن. لا يمكن للكائنات الأخرى أو العالم الخارجي الوصول إليها إلا من خلال إجراءات محددة جيدا : واجهة الكائن.

إن استخدام الأصناف في برامجك يسمح لك - من بين الفوائد الأخرى - بتجنب استخدام الحد الأقصى من المتغيرات العالمية. يجب أن تعرف أن استخدام المتغيرات العالمية أمر خطير جدا، حتى أنه أكثر أهمية من البرامج كبيرة الحجم، لأنه من الممكن

دائما تغيير هذه المتغيرات، أو حتى إعادة تعريفها، في أي جزء من البرنامج (هذا الخطر يزداد سوءاً إذا كان هنالك العديد من المبرمجين يعملون على نفس البرنامج).

الفائدة الثانية الناتجة عن استخدام الأصناف هي إمكانية بناء كائنات جديدة من الكائنات الموجودة، وبالتالي إعادة استخدام المساحات البرمجية الكبيرة المكتوبة بالفعل (بدون لمسها !)، لاشتقاق ميزة جديدة. هذا سيكون ممكنا بفضل مفاهيم الاشتقاق وتعدد الأشكال :

- الاشتقاق هي آلية تسمح لك ببناء صنف جديد "ابن أو طفل" من صنف "أم أو الأصل". الطفل سيرث جميع خصائص وجميع وظائف الأم، ويمكنك إذًا إضافة ما تريد عليها.
- يمكن لتعدد الأشكال تعيين سلوكيات مختلفة للكائنات المشتقة من بعضها البعض، أو من نفس الكائن أو من وفق لسياق معين .

قبل المضي قدما، لاحظ هنا أن البرمجة الشيئية هي شيء اختياري في بيثون. يمكنك تطوير العديد من البرامج بدون استخدامها، مع أدوات أكثر بساطة من دالات. اعلم أنك إذا بذلت المزيد من الجهد في تعلم البرمجة بمساعدة الأصناف، سوف تتقن مستوى أعلى، مما يسمح لك التعامل مع مشاكل أكثر تعقيدا. وبعبارة أخرى، سوف تصبح مبرمجا متخصصًا أكثر. لإقناعك بذلك، تذكر التقدم الذي قمت به خلال هذه الدورة :

- في بداية دراستك، بدأت باستخدام تعليمات بسيطة. لقد كنت إلى حد ما "مبرمج باليد" (و هذا معناه تقريبا دون أدوات).
- ثم تعرفت على الدالات التي تم تعريفها مسبقا (انظر للفصل السادس)، وتعلمت أن هناك مجموعات واسعة من الأدوات المتخصصة التي قدمت من قبل مبرمجين أخرين.
- تعلمت كتابة دالات خاصة بك (انظر للفصل السابع وما بعده)، فأصبحت قادراً على صنع أدوات جديدة خاصة بك، وهذه يمنحك تحكمًا كبيراً إضافيا.
- إذا كنت ستبدأ الآن برمجة الأصناف، سوف تتعلم كيفية بناء آلات لإنتاج الأدوات. وهذا من الواضح أكثر تعقيدا من صنع الأدوات مباشرة، ولكن هذه تفتح لك أبواب أوسع من ذلك بكثير !

فهم هذه الأصناف تساعدك على السيطرة على مجال واجهات المستخدم الرسومية (tkinter، wxPython) وإعدادك للتصدي بشكل فعال على لغات حديثة أخرى مثل سي بلس بلس أو الجافا .

تعريف صنف أولي

لإنشاء صنف كائن بيثون، نستخدم العبارة **class**. سوف نتعلم استخدام هذه العبارة، بدءا من تعريف نوع كائن أساسي والذي سيكون ببساطة مجرد نوع من البيانات الجديدة. ولقد استخدمنا أنواعا مختلفة من البيانات حتى الآن، ولكن كانت كل مرة من نوع مدمج في اللغة نفسها. سوف نقوم الآن يصنع نوع مركب جديد : النوع **Point**.

هذا النوع يتوافق مع مفهوم هندسة مستوى النقطة. في المستوى، يتم تمييز النقطة من رقمين (الإحداثيات x و y). في التدوين الرياضي، يتم إذا تعريف النقطة من خلال إحداثيات x و y في زوج من الأقواس. على سبيل المثال النقطة (25, 17). وهنالك طريقة طبيعية لتمثيل النقطة في بيثون ليتم استخدامها لإحداثيات قيمتين من نوع حقيقي. لكن نحن نريد الجمع بين هاتين القيمتين في كيان واحد، أو في كائن واحد. لتحقيق ذلك سوف نقوم بتعريف الصنف **Point()** :

```
>>> class Point(object):
...     "Définition d'un point géométrique"
```

تعريف الأصناف يمكن أن يكون في أي مكان في البرنامج، لكن يتم وضعها بشكل عام في البداية (أو في وحدة يتم استدعاؤها). المثال أعلاه هو على الأرجح الأبسط الذي يمكن تخيله. سطر واحد يكفي لتعريف نوع كائن جديد **Point()**.

نلاحظ ما يلي :

- العبارة **class** هو مثال آخر على العبارة المجمعة. لا تنسَ النقطتين في نهاية السطر فهي إلزامية، ومسافة البادئة في كتلة التعليمات التي تليها. هذا المجمع يجب أن يحتوي على الأقل على سطر واحد. في مثالنا المبسط للغاية، هذا السطر لا يحتوي سوى على تعليق بسيط. كما رأينا سابقا في الدالات (انظر الصفحة 74)، يمكنك إضافة سلسلة نصية مباشرة بعد العبارة **class**، من أجل وضع تعليق لإدراجها تلقائيا إلى الوثائق الداخلية لبيثون. تعود دائما على وضع سلسلة لوصف الصنف هنا .
- تم وضع الأقواس ليحتوي على مرجع صنف موجود. هذا الأمر مطلوب للسماح لآلية الميراث. كل الأصناف الجديدة التي نصنعهايمكن أن ترث من الصنف الأصل مجموعة من المميزات، التي تضاف إليها تلقائيا. عندما نريد إنشاء صنف أساسي (و هذا معناه أنه غير معتمد على أي صنف آخر، كما في مثالنا الصنف **Point()** - جب أن يكون مرجع الإشارة حسب الاتفاقية هو الاسم الخاص **object**، مما يعني أنه جد جميع الأصناف الأخرى.[65]
- الاتفاقية المنتشرة بكثرة هي أن يتم إعطاء أسماء للأصناف تبدأ بحرف كبير. في بقية النص، سوف نحترم هذه الاتفاقية، والاخر الذي يطلب في السرد، ويرتبط مه كل اسم صنف زوج من الأقواس، كما نفعل بالأسماء الدالات .

65عند تعريف صنف أساسي، يمكنك حذف الأقواس والمرجع إلى الصنف كائن السلف (الذي قبله)؛ هذه المؤشرات أصبحت اختيارية في البايثون 3. سنواصل إستخدامها في هذا النص، لكن من الجيد أن نقوم بتوضيح أهمية مفهوم الوراثة .

نحن نريد إذًا تعريف الصنف **Point**(). يمكننا الآن صنع كائنات من هذا الصنف، والتي نسميها أيضا مثيل للصنف. وتسمى هذه العملية بالتمثيل. على سبيل المثال ننشئ كائنا جديدا **p9**[66] :

```
>>> p9 = Point()
```

بعد هذه العبارة المتغير **p9** يحتوي على مرجع للكائن الجديد **Point**(). يمكننا أن نقول أن **p9** هو مثيل جديد للصنف **Point**().

> تنبيه
>
> مثل الدالات، يجب على الأصناف التي يتم إستدعاؤها في التعليمة أن تكون مصحوبة بقوسين (حتى لو لم يتم تمرير برامتر). سوف نرى في الواقع أن الأصناف يمكن استدعاؤها مع برامترات .

دعونا نرى ما إذا كنا نستطيع أن نفعل شيئا مع الكائن الجديد **p9** :

```
>>> print(p9)
<__main__.Point object at 0xb76f132c>
```

الرسالة التي تم إرجاعها بواسطة بيثون، سوف تفهم على الفور أن **p9** هو مثيل للصنف **Point**()، والتي تم تعريفها جيدا في المستوى الرئيسي (main) للبرنامج. وقد تم وضعها في موقع محدد في الذاكرة الحية (ram)، والتي تظهر الآن بالترقيم السداسي العشري .

```
>>> print(p9.__doc__)
Définition d'un point géométrique
```

كما شرحنا الدالات (انظر للصفحة 74)، يتم ربط سلاسل توثيق كائنات بيثون المختلفة مع سمة المعرفة **__doc__**. وإذا فمن الممكن دائما العثور على الوثائق المرتبطة مع أي كائن بيثون، نت خلال استدعاء هذه السمة.

سمات (أو متغيرات) المثيل

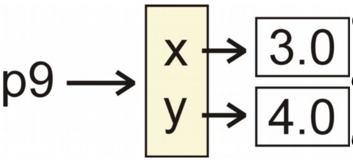
الكائن الذي صنعناه هو مجرد صَدَفَة فارغة. سنضيف الآن عناصره، بتعيين بسيط، باستخدام نظام تأهيل الأسماء بالنقاط[67] :

[66] في بيثون، يمكننا إنشاء مثيل كائن بمساعدة تعليمة بسيطة خاصة. في اللغات الأخرى تتطلب استخدام تعليمة خاصة، مثل `new` ليبين أن يقوم بإنشاء كائن جديد من العفن. على سبيل المثال: `p9 = new Point()`.

[67] هذا نظام الترقيم مشابه للذي استخدمناه لوصف متغير لوحدة، مثل `math.pi` أو `string.ascii_lowercase`. و سوف نعود في وقت لاحق، ولكن نعرف الآن أن الوحدات يمكنها أن تحتوي على دالات، ولكن حتى أصناف ومتغيرات. حاول على سبيل المثال :

```
>>> import string
>>> string.capwords
>>> string.ascii_uppercase
>>> string.punctuation
>>> string.hexdigits
```

```
>>> p9.x = 3.0
>>> p9.y = 4.0
```



المتغيران **x** و **y** التي قمنا بتعريفهم من خلال الربط المباشر مع **p9** ، هم الآن سمات للكائن **p9**. يمكن أيضا استدعاء متغيرات المثيل. في الواقع يتم إدراجها، أو تغليفها في هذا المثيل (أو الكائن). الرسم البياني على اليمين يظهر نتيجة هذه التعيينات : المتغير **p9** يحتوي على مرجع يشير إلى موقع الكائن الجديد في الذاكرة، والذي يحتوي على سمتين **x** و **y**. وهذه تتضمن مراجع للقيم 3.0 و 4.0 المخزنة في أماكن أخرى.

يمكننا استخدام سمات كائن في أي تعبير، تماما مثل أي متغير عادي :

```
>>> print(p9.x)
3.0
>>> print(p9.x**2 + p9.y**2)
25.0
```

بسبب التغليف في الكائن، السمات هي متغيرات مستقلة عن المتغيرات الأخرى التي قد تحمل نفس الاسم. على سبيل المثال، التعليمة `x = p9.x` معناه : "استخرج من مرجع الكائن في **p9** قيمة السمة **x**، وقم بربط هذه القيمة بالمتغير **x**". ولا يوجد تعارض بين المتغير المستقل **x**. وبين السمة **x** للكائن **p9**. الكائن **p9** يحتوي على مساحة الأسماء خاصة به، مستقلة عن مساحة أسماء الرئيسية التي تجد المتغير **x**.

> هام / الأمثلة الموجودة هنا مؤقتة .
>
> لقد رأينا أنه من السهل جدا إضافة سمة إلى كائن باستخدام تعليمة بسيطة مثل *p9.x = 3.0* وهذا يمكن تحمله في بيثون (و هذه نتيجة لطبيعة الحيوية لبيثون). ولكن لا ينصح بذلك حقا. كما سوف تفهم لاحقا . فنحن إذا لن نستخدم هذا النهج سوى للتوضيح. وفقط من أجل تبسيط الشرح لدينا لسمات المثيل. وستم تطوير الطريقة الصحيحة في الفصل القادم .

تمرير كائن ك برامتر عند استدعاء دالة

الدالات يمكنها استخدام الكائنات كبرامترات، ويمكن أيضا استخدام الكائن كقيمة للعودة. على سبيل المثال، يمكنك تعريف دالة مثل هذه :

```
>>> def affiche_point(p):
...     print("coord. horizontale =", p.x, "coord. verticale =", p.y)
```

البرامتر **p** الذي يستخدم من قبل هذه الدالة يجب أن يكون كائن من نوع **Point()**. الذي يستخدم متغيرات المثيل **p.x** و **p.y**. عندما تستدعى هذه الدالة، يجب عليك إذا توفير كائن من نوع **Point()** كبرامتر. جرب مع الكائن **p9** :

```
>>> affiche_point(p9)
coord. horizontale = 3.0 coord. Verticale = 4.0
```

تمرين

1.11 اكتب دالة **distance()** تسمح لك بحساب المسافة بين نقطتين. (يجب أن تتذكر نظرية فيثاغورس !) هذه الدالة تنتظر كائنين من نوع **Point()** كبرامتر .

التشابه والتفرد

في اللغة التي نتحدث بها، نفس الكلمة يمكن أن يكون لها معان مختلفة اعتمادا على السياق الذي تستخدم فيه. والنتيجة يمكن أن نفهم بعض عبارات التي تستخدم فيها هذه الكلمات بمعان مختلفة (مصطلحات غامضة).

على سبيل المثال كلمة "نفس - même" لديها معان كثيرة في الجمل : "شارلز وأنا لدينا نفس السيارة" و"شارلز وأنا لدينا نفس الأم". في الجملة الأول، ما أعنيه أنني وشارلز لدينا نفس نموذج السيارة لكنهما سيارتين مختلفتين. وفي الجملة الثانية، ما أعنيه أنني وشارلز لدينا نفس الأم (نفس الشخص).

عندما نتعامل مع كائنات البرامج، يمكن أن تجد نفس الغموض. على سبيل المثال، فإذا تحدثنا عن المساوات بين كائنين **Point()**، هذا معناه هذان الكائنات يحتويان على نفس المعطيات (سماتهم)، أو يعني هذا أننا نتحدث عن مرجعين لنفس الكائن ؟ على سبيل المثال، التعليمات التالية :

```
>>> p1 = Point()
>>> p1.x = 3
>>> p1.y = 4
>>> p2 = Point()
>>> p2.x = 3
>>> p2.y = 4
>>> print(p1 == p2)
False
```

هذه التعليمات تقوم بإنشاء كائنين **p1** و **p2** يبقون مستقلين، حتى لو كانوا ينتمون إلى نفس الصنف وكانوا لديهم نفس المحتويات. التعليمة الأخيرة تجرب مساواة هذين الكائنين (علامة مساوات مزدوجة)، والنتيجة **False** (سالبة) : إذا لا يتساوون.

يمكننا تأكيد هذا بطريقة أخرى :

```
>>> print(p1)
<__main__.Point instance at 00C2CBEC>
>>> print(p2)
<__main__.Point instance at 00C50F9C>
```

معلومة واضحة : المتغيرين **p1** و **p2** مراجعهم مختلفة، تم تخزينهم في أماكن مختلفة في ذاكرة الحاسوب.

حاول شيئا آخر الآن :

```
>>> p2 = p1
>>> print(p1 == p2)
True
```

بناء على التعليمة p2 = p1 ، قمنا بتعيين محتوى **p1** إلى **p2**. وهذا معناه متغيرين يشيران إلى مرجع الكائن نفسه **p1** و **p2** هي أسماء مستعارة[68] لبعضها البعض.

اختبار المساواة هذه المرة أرجع لنا القيمة **True**، (صحيح)، وهذا معناه أن اختبار القوسين صحيح: **p1** و **p2** تم تعيينهم إلى كائن واحد، إذا لم تقتنع حاول:

```
>>> p1.x = 7
>>> print(p2.x)
7
```

عندما نغير سمة **x** لـ **p1**، نلاحظ أن سمة **x** لـ **p2** تتغير أيضا..

```
>>> print(p1)
<__main__.Point instance at 00C2CBEC>
>>> print(p2)
<__main__.Point instance at 00C2CBEC>
```

المرجعين **p1** و **p2** يشيران إلى نفس المكان في الذاكرة.

كائنات تتكون من كائنات

الآن لنفترض أننا نريد تعريف دالة تمثل مستطيلات. ببساطة، نحن نعتبر أن هذه المستطيلات سوف تكون دائما موجهة أفقيا أو عموديا، أما قطريا فهذا مستحيل.

ما هي المعلومات التي نحتاجها لتعريف هذه المستطيلات ؟

هنالك العديد من الاحتمالات. يمكننا على سبيل المصال تحديد مكان وسط المستطيل (احداثيتان) وتحديد حجمها (الطول والعرض). ويمكننا أيضا تحديد أماكن الزوايا أعلى اليسار وأدنى اليمين أو يمكننا تحديد مكان زاوية أعلى اليسار والحجم. سوف نستخدم الطريقة الأخيرة.

عرف إذا صنف الخاص بنا الجديد :

```
>>> class Rectangle(object):
        "définition d'une classe de rectangles"
```

و سوف تخدمنا الآن لإنشاء مثيل :

68بشأن ظاهرة الأسماء المستعارة. انظر أيضا إلى صفحة Error: Reference source not found.

```
>>> boite = Rectangle()
>>> boite.largeur = 50.0
>>> boite.hauteur = 35.0
```

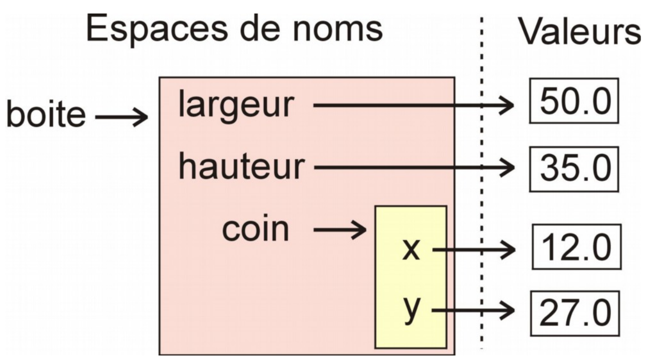
لقد صنعنا الكائن الجديد **Rectangle()** وأعطيناه سمتين. لتعريف الزاوية أعلى اليسار، سوف نستخدم مثيل جديد للصنف **Point()** الذي عرفناه سابقا. وبالتالي فإننا أنشأنا كائنا جديدا، داخل كائن آخر!

```
>>> boite.coin = Point()
>>> boite.coin.x = 12.0
>>> boite.coin.y = 27.0
```

في أول هذه التعليمات الثلاث، قمنا بصنع سمة جديدة **coin** للكائن **boite**. ثم، للوصول لهذا الكائن والذي هو في حد ذاته موجود في كائن آخر، استخدمنا تصنيف الأسماء الهرمي (بمساعدة النقاط) والذي تحدثنا عنه عدة مرات سابقا.

التعبير `boite.coin.y` يعني "اذهب إلى مرجع الكائن في المتغير **boite**. وفي هذا الكائن، ثم جد السمة **coin**، ثم اذهب إلى مرجع كائن في هذه السمة. فإذا وجد الكائن، قم بتحديد سمته **y**".

قد تفهم أفضل إذا كان كل هذا في رسم تخطيطي، مثل هذا :



الأسم **boite** يوجد في مساحة الأماكن الرئيسة. وهو يشير إلى مساحة أخرى للأسماء محجوز للكائن المقابل، في هذه المساحة يتم حفظ أسماء **hauteur**، **largeur** و **coin**. وهذه تشير إلى مساحات أخرى للأسماء (مثل الاسم « **coin** »)، أو إلى قيم تم تعريفها جيدا، والتي يتم تخزينها في أماكن أخرى.

بيثون يقوم بحجز مساحات لأسماء مختلفة لكل وحدة، كل صنف، كل مثيل وكل كائن. يمكنك الاستفادة من كل هذه الأماكن المجزأة بشكل جيد لتحقيق برامج قوية، وهذا يعني برامج لا يمكن أن تتداخل بسهولة .

الكائنات مثل القيم رجوع الدالة

لقد رأينا أعلاه أن الدالات يمكنها استخدام الكائنات كبرامترات. ويمكنها أن تمرر مثيل كقيمة رجوع. على سبيل المثال، الدالة **trouveCentre()** في الأسفل يمكن استدعاؤها مع برامتر من نوع **Rectangle()** وهي ترسل كائنا من نوع **Point()**، والتي تحتوي على إحداثيات وسط المستطيل .

```
>>> def trouveCentre(box):
...     p = Point()
...     p.x = box.coin.x + box.largeur/2.0
...     p.y = box.coin.y + box.hauteur/2.0
...     return p
```

على سبيل المثال، يمكنك استدعاء هذه الدالة، باستخدام برامتر الكائن **boite** الذي تم تعريفه بالأعلى :

```
>>> centre = trouveCentre(boite)
>>> print(centre.x, centre.y)
37.0  44.5
```

تعديل الكائنات

يمكننا تعديل خصائص كائن عن طريق تعيين قيم جديدة لسماته. على سبيل المثال، يمكننا تعديل حجم المستطيل (دون تغيير موقعه)، عن طريق إعادة تعيين سماتيه hauteur و largeur :

```
>>> boite.hauteur = boite.hauteur + 20
>>> boite.largeur = boite.largeur – 5
```

يمكننا فعل هذا في بيثون، لأنه في هذه اللغة خصائص الكائن دائما عامة (على الأقل حتى الإصدار الحالي : 3.1). في اللغات الأخر يوجد فرق واضح بين السمات العامة (يمكن الوصول إليها من خارج الكائن) والسمات الخاصة (الذي يمكن الوصول إليها فقط عن طريق الخوارزمية المدرجة في الكائن نفسه).

و لكن كما ذكرنا في الأعلى (حول تعريف سمات تعيين بسيطة من خارج الكائن)، طريقة تعديل سمات مثيل ليس موصى به، لأنه ينقص أحد الأهداف الأساسية للبرمجة الشيئية، والذي يهدف إلى إنشاء فصل صارم بين وظائف الكائن (حتى التي تم تعريفها في الخارج) وكيف يتم تنفيذها في الواقع هذه الوظيفة في الكائن (و العالم الخارج لا يعرف).

عمليا، هذا يعني أننا يجب أن نتعلم كيفية عمل تشغيل الكائنات باستخدام أدوات مناسبة، والتي نسميها الأساليب.

و الآن، بعد أن نفهم هذه العملية، يمكننا تحديد قاعدة بعدم تغيير سمات الكائن عن طريق العالم الخارجي مباشرة كما فعلنا الآن. وبدلا من ذلك نحن نريد استخدام هذا الأسلوب لإعداده خصيصا لهذا الغرض، كما سوف نشرحه في الفصل القادم. جمع هذه الأساليب يكون لنا ما يسمى بواجهة الكائن .

12

الأصناف والأساليب والميراث

الأصناف التي عرفناها في الفصل السابق يمكن اعتبارها كمساحات لأسماء معينة، وفيها نحن لن نضع حتى الآن سوى المتغيرات (سمات المثيل). ويجب الآن إعطاء وظيفة لهذه الأصناف .

الفكرة الأساسية للبرمجة الشيئية هي في الواقع جمع الكائنات في نفس المجموعة معا، في كل مرة عدد من البيانات (سمات المثيل)، والخوارزميات تقوم بمختلف المعالجات على هذه البيانات (و هي الأساليب، أي دالات معينة مغلقة في كائن) .

الكائن = [سمات + أساليب]

و بهذه الطريقة يتم جمع خصائص الكائن والدالات في نفس "الكبسولة" التي تعمل عليها، يتوافق مع مصممي البرامج رغبة في بناء كيانات حاسوبية ذات سلوك مشابه للكائنات في العالم الحقيقي الذي يحيط بنا.

على سبيل المثال الودجة "زر" في تطبيق رسومي. فإنه يبدو من المعقول أن تتمنى أن الكائن الحاسوبي الذي ندعو سلوكه الذي يشبه أي زر جهاز في العالم الحقيقي. ونحن نعرف عمل الزر الحقيقي (قادر على غلق أو فتح الدارة الكهربائية) هي مدمجة في الكائن نفسه (و كذلك خصائصه الأخرى مثل حجمه ولونه وإلخ ...) .و بنفس الطريقة، نحن نأمل إذا اختلاف خصائص زر البرنامج (حجمه، مكانه، لونه، النص المكتوب عليه)، ولكن أيضا تعريف ما يحدث عندما القيام بحركات بالفأرة على هذا الزر، سواء أن يتم جمع داخل كيان محدد جدا في البرنامج، حتى لا يكون هنالك أي خلط بين الأزرار، أو حتى أكثر من ذلك بين زر والكيانات الأخرى .

تعريف أسلوب

لتوضيح كلامنا، سوف نعرف صنفا جديدا **Time()**، والذي ينبغي له أن يسمح لنا بأداء مجموعة من العمليات على اللحظات والمدد (جمع مدة)، إلخ :

```
>>> class Time(object):
...     "définition d'objets temporels"
```

اصنع الآن كائنا من نفس النوع، وأضف له متغيرات المثيل لحفظ الساعات والدقائق والثواني :

```
>>> instant = Time()
>>> instant.heure = 11
>>> instant.minute = 34
>>> instant.seconde = 25
```

باعتباره تمرينا، اكتب الآن بنفسك دالة **affiche_heure()** ، والتي تستخدم لعرض محتويات كائن من صنف **Time()** في الشكل التقليدي "ساعات:دقائق:ثواني". عند تطبيق الكائن الذي قمنا بصنعه فوق، هذه الدالة يجب أن تعرض : 11:34:25

```
>>> affiche_heure(instant)
11:34:25
```

دالتك قد يكون شكلها كهذا :

```
>>> def affiche_heure(t):
...     print(str(t.heure) +":" +str(t.minute) +":" +str(t.seconde))
```

أو أفضل من ذلك، مثل هذا :

```
>>> def affiche_heure(t):
...     print("{0}:{1}:{2}".format(t.heure, t.minute, t.seconde))
```

بتطبيق تقنية تنسيق السلاسل التي شرحناها في الصفحة 149.

ربما تكون بحاجة إلى استخدام كائنات الصنف **Time()**، هذه الدالة ربما تكون مفيدة لك للعرض.

قد يكون من الحكمة تغليف هذه الدالة **affiche_heure()** في الصنف **Time()** -نفسها، بطريقة نجعلها دائما متوفرة تلقائيا، كلما أردنا معالجة كائنات من الصنف **Time()**.

الدالة التي نريد تغليفها في صنف يسمى بشكل تفضيلي أسلوب méthode.

و لقد تحدثنا عن الأساليب في عدة مرات في الفصول السابقة من هذا الكتاب، وأنت تعرف بالفعل أن الأساليب هي دالات مرتبطة بصنف محدد بكائنات. بقي فقط أن تعلم كيفية صنع هذه الدالة .

تعريف محدد لأسلوب في سكريبت

تعريف أسلوب مثل تعريف دالة، هذا معناه كتابة كتلة من التعليمات بعد الكلمة المحجوز **def**، لكن مع اختلافين:

- تعريف الأسلوب يكون مكانه دائما داخل تعريف الصنف، بطريقة حتى يظهر بوضوح العلاقة بين الأسلوب والصنف.
- تعريف الأسلوب يجب أن يحتوي دائما على برامتر واحد على الأقل، والتي يجب أن تكون مرجع المثيل، وهذا البرامتر يجب أن يكون دائما الأول .

يمكنك مبدئيا استخدام أي اسم متغير للبرامتر الأول، لكن يوصى بشدة متابعة الاتفاقية لذا يكون دائما اسمه: **self**.

البرامتر **self** ضروري، لأنه يجب أن تكون قادرا على تحديد المثيل الذي سيتم ربطه، في جزء تعليمات من تعريفه. سوف تفهم هذا بأكثر سهولة مع الأمثلة القادمة.

> نلاحظ أن تعريف الأسلوب يحتوي دائما على برامتر واحد : *self*. في حين أنه يمكن تعريف الدالة بدون أن تحتوي على أي شيء .

انظر كيف يحدث هذا عمليا :

للتأكد من أن الدالة **affiche_heure()** ستكون أسلوبا للصنف **Time()**، سوف نحرك ببساطة تعريفها إلى داخل الصنف :

```
>>> class Time(object):
...     "Nouvelle classe temporelle"
...     def affiche_heure(t):
...         print("{0}:{1}:{2}".format(t.heure, t.minute, t.seconde))
```

من الناحية الفنية، فإن هذا يكفي تماما، لأن البرامتر t تعين المثيل للسمات المرفقة **minute**، **heure** و **seconde**. نظرا لدورها، فمن المستحسن تغيير اسمها إلى **self** :

```
>>> class Time(object):
...     "Nouvelle classe temporelle"
...     def affiche_heure(self):
...         print("{0}:{1}:{2}".format(self.heure, self.minute, self.seconde))
```

تعريف الأسلوب **affiche_heure()** أصبح الآن جزءا من كتلة التعليمات البادئة التالية للتعليمة الصنف (و هو أيضا جزء من الوثائق "Nouvelle classe temporelle") .

اختبار الأسلوب، في أي مثيل

لدينا الآن الصنف **Time()**، مع الأسلوب **affiche_heure()**. من حيث المبدأ، نحن الآن قادرون على صنع كائنات من هذا الصنف، وتطبيق عليها هذا الأسلوب. دعونا نرى ما إذا كان يعمل. وللقيام بذلك، نبدأ من تمثيل كائن:

```
>>> maintenant = Time()
```

إذا حاولنا بسرعة اختبار الأسلوب الجديد على هذا الكائن، فإنه لا يعمل :

```
>>> maintenant.affiche_heure()
AttributeError: 'Time' object has no attribute 'heure'
```

هذا أمر طبيعي : لم نصنع بعد سمات المثيل. يجب أن نقوم على سبيل المثال :

```
>>> maintenant.heure = 13
>>> maintenant.minute = 34
>>> maintenant.seconde = 21
```

... ونحاول مرة أخرى. وهذه المرة إشتغل :

```
>>> maintenant.affiche_heure()
13:34:21
```

في عدة مرات، لقد ذكرنا أنه ليس من المستحسن إنشاء سمات مثيل لربطها مباشرة من خارج الكائن نفسه. ومن المضايقات الأخرى، فإن هذا يؤدي إلى أخطاء أخرى مثل التي ذكرناها سابقا. دعونا نرى الآن القيام بعمل أفضل .

الأسلوب المنشئ

الخطأ الذي تحدثنا عنه في الفقرة السابقة هل يمكن تجنبه؟

هذا لا يحدث فعليا، فإذا قمنا بتنظيم الأسلوب **affiche_heure()** لجعله دائما يعرض شيئا ما، دون أي يكون من الضروري تعديل الكائن المنشئ حديثا. وبعبارات أخرى، سيكون من الحكمة أن متغيرات المثيل تكون معرفا مسبقا في البداية الصنف، مع قيمة افتراضية.

لتحقيق هذا، سوف نقوم باستدعاء أسلوب معين، والذي سوف يعين فيما بعد تحت اسم المنشئ. الأسلوب المنشئ لديه طريقة فريدة لكي يتم تشغيله تلقائيا عند تمثيل كائن جديد بداية من الصنف. لذا يمكننا وضع كل ما يبدو ضروريا لتهيئته تلقائيا عند إنشاء كائن.

و هو معروف عند بيثون، الأسلوب المنشئ يدعى ضروري بـ **__init__** (رمزي "في أسفل السطر"، وكلمة **init** ثم رمزي "في أسفل السطر) .

مثال :

```
>>> class Time(object):
...     "Encore une nouvelle classe temporelle"
...     def __init__(self):
...         self.heure =12
...         self.minute =0
...         self.seconde =0
...     def affiche_heure(self):
...         print("{0}:{1}:{2}".format(self.heure, self.minute, self.seconde))
```

كما كان سابقا، إنشاء كائن من هذا الصنف وتجربة الأسلوب **affiche_heure()** :

```
>>> tstart = Time()
>>> tstart.affiche_heure()
12:0:0
```

لم نحصل على أي خطأ هذه المرة. في الواقع : عندما يتم إنشاء مثيل. الكائن **tstart** يتم تعيين 3 سمات، **heure** و **minute** و **seconde** بواسطة الأسلوب المنشئ مع 12 وصفر كقيمة أولية. ولهذا كائن من هذا الصنف موجود، يمكن أن نطلب عرض هذه السمات فورا.

و الاستفادة من هذه التقنية يزداد وضوحا إذا أضفنا شيئا آخر.

مثل أي أسلوب يجب احترامه، الأسلوب **__init__()** تكون برامتراته جاهزة. وفي حالة هذا الأسلوب الخاص من المنشئ البرامترا يمكن أن تلعب دورا مثيرا للاهتمام، لأنها سوف تسمح بتهيئة بعض متغيرات التمثيل في وقت تمثيل الكائن.

يرجى إذا الرجوع إلى التمرين السابق، وغير تعريف الأسلوب **__init__()** على النحو التالي ::

```
...     def __init__(self, hh =12, mm =0, ss =0):
...         self.heure =hh
...         self.minute =mm
...         self.seconde =ss
```

أسلوبنا الجديد **__init__()** لديه الآن 3 برامترات، ولكل منها قيمة افتراضية. وبالتالي نحصل على صنف أكثر تقدما. عندما كنا نمثل كائن من هذا الصنف، يمكننا الآن تهيئة سماته الرئيسية بمساعدة البرامترات، ضمن تعليمات التمثيل. وإذا أردنا حذف كل أو جزء منها، السمات تقوم بتسليم بأي شكل من الأشكال القيم الافتراضية.

عندما نكتب تعليمات لتمثيل كائن جديد، وعندما تريد تمرير برامترات لأسلوب المنشئ يكفي أن تضعهم بين قوسين تصاحب اسم الصنف. لنشرع إذا بالضبط مثل طريقة عندما نريد استدعاء أي دالة.

هذا مثال لصنع وتمثيل كائن جديد **Time()** في نفس الوقت :

```
>>> recreation = Time(10, 15, 18)
>>> recreation.affiche_heure()
10:15:18
```

منذ الآن متغيرات التمثيل لها الآن قيم افتراضية، نحن نستطيع أيضا صنع قيم افتراضية للكائن **Time()** بحذف واحد أو أكثر من برامتر :

```
>>> rentree = Time(10, 30)
>>> rentree.affiche_heure()
10:30:0
```

أو أكثر :

```
>>> rendezVous = Time(hh =18)
>>> rendezVous.affiche_heure()
18:0:0
```

تمارين

1.12 عرف الصنف **Domino()** الذي يسمح بتمثيل كائنت لمحاكات أجزاء للعبة الدومينو. المنشئ في هذا الصنف يهيئ قيم هذه النقاط على جانبي A و B للدومينو (القيم الافتراضية = 0). وقم بتعريف أسلوبين آخرين :

- أسلوب **affiche_points()** الذي يعرض نقاط على الجانبين;
- أسلوب **valeur()** الذي يقوم بإرجاع مجموع النقاط الموجودة على الجانبين ..

أمثلة استخدام لهذا الصنف :

```
>>> d1 = Domino(2,6)
>>> d2 = Domino(4,3)
>>> d1.affiche_points()
face A : 2  face B : 6
>>> d2.affiche_points()
face A : 4  face B : 3
>>> print("total des points :", d1.valeur() + d2.valeur())
15
>>> liste_dominos = []
>>> for i in range(7):
...     liste_dominos.append(Domino(6, i))
>>> print(liste_dominos[3])
<__main__.Domino object at 0xb758b92c>
```

إلخ.

2.12 عرف الصنف **CompteBancaire()**, الذي يسمح بتمثيل كائنات مثل **compte1**, **compte2**, إلخ. منشئ هذا الصنف يهيئ سمتي تمثيل **nom** و **solde**, مع قيم افتراضية 'Dupont' و 1000.

و الثلاثة أساليب الأخرى التي يجب تعريفها :

- **depot(somme)** يسمح بإضافة كمية معينة على الدفع

- **retrait(somme)** يسمح بإزالة كمية معينة على الدفع
- **affiche()** يسمح بعرض اسم صاحب الحساب ورصيد حسابه .

أمثلة استخدام لهذا الصنف :

```
>>> compte1 = CompteBancaire('Duchmol', 800)
>>> compte1.depot(350)
>>> compte1.retrait(200)
>>> compte1.affiche()
Le solde du compte bancaire de Duchmol est de 950 euros.
>>> compte2 = CompteBancaire()
>>> compte2.depot(25)
>>> compte2.affiche()
Le solde du compte bancaire de Dupont est de 1025 euros.
```

3.12 عرف الصنف **Voiture()** الذي يسمح بتمثيل كائنات استنساخ سلوك السيارات. المنشئ لهذا الصنف يهيئ سمات التمثيل التالية، مع القيم الافتراضية المبينة :

`marque = 'Ford',  couleur = 'rouge',  pilote = 'personne',  vitesse = 0.`

عند إنشاء مثيل لكائن جديد **Voiture()**، يمكننا اختيار شركته ولونه لكن ليس سرعته وليس اسمه وليس اسم سائقها.

سيتم تعريف هذه الأساليب :

- **choix_conducteur(nom)** يسمح بتحديد (أو تغيير) اسم السائق.
- **accelerer(taux, duree)** يسمح بتبديل سرعة السيارة. تبديل السرعة يساوى حسب سرعة المنتج : المعدل × المدة. على سبيل المثال، إذا كان تسارع السيارة بمعدل 1,3 متر في الثانية لمدة 20 ثانية، سوف تحصل على سرعة 26 متر في الدقيقة. ويسمح بالمعدلات السلبية (و هو التباطؤ). الاختلاف في السرعة لم يسمح له إذا كان السائق "شخص" .
- **affiche_tout()** يسمح بإظهار خصائص الموجودة في السيارة، وهذا معناه شركتها، لونه واسم سائقها وسرعتها.

أمثلة استخدام لهذا الصنف :

```
>>> a1 = Voiture('Peugeot', 'bleue')
>>> a2 = Voiture(couleur = 'verte')
>>> a3 = Voiture('Mercedes')
>>> a1.choix_conducteur('Roméo')
>>> a2.choix_conducteur('Juliette')
>>> a2.accelerer(1.8, 12)
>>> a3.accelerer(1.9, 11)
Cette voiture n'a pas de conducteur !
>>> a2.affiche_tout()
Ford verte pilotée par Juliette, vitesse = 21.6 m/s.
>>> a3.affiche_tout()
Mercedes rouge pilotée par personne, vitesse = 0 m/s.
```

4.12 عرف الصنف **Satellite()** الذي يسمح بتمثيل كائنات تحاكي الأقمار الصناعية التي تطلق في الفضاء حول الأرض. المنشئ لهذا الصنف يهيئ سمات التمثيل التالية، مع القيم الافتراضية المبينة :

```
masse = 100,  vitesse = 0.
```

عند إنشاء مثيل لكائن جديد **Satellite()**، يمكن اختيار اسمه وكتلته وسرعته.

الأساليب التالية سوف يتم تعريفها :

- **impulsion(force, duree)** تسمح بتفاوت سرعة القمر الصناعي. لمعرفة كيفية فعل هذا، تذكر دروس الفيزياء : سرعة التفاوت **Δv** التي يمر بها الكائن بكتلة **m** تخضع لحركة قوة **F** في غضون وقت vaut **t** $\delta(ك)v=\frac{F\times t}{m}$. على سبيل المثال : قمر صناعي يزن 300 كيلوغرام يخضع لقوة 600 نيوتن لمدة 10 ثواني سوف تزداد سرعته(أو تنقص) 20م/ثانية.
- **affiche_vitesse()** عرض اسم القمر الصناعي وسرعته الحالية.
- **energie()** تقوم بإرجاع إلى البرنامج الذي استدعاؤها قيمة الطاقة الحركية للقمر الصناعي. .

تذكير / يتم حساب الطاقة الحركية باستخدام الصيغة : $E_c=\frac{m\times v^2}{2}$

أمثلة استخدام لهذا الصنف :

```
>>> s1 = Satellite('Zoé', masse =250, vitesse =10)
>>> s1.impulsion(500, 15)
>>> s1.affiche_vitesse()
vitesse du satellite Zoé = 40 m/s.
>>> print (s1.energie)
200000
>>> s1.impulsion(500, 15)
>>> s1.affiche_vitesse()
vitesse du satellite Zoé = 70 m/s.
>>> print (s1.energie)
612500
```

مساحات أسماء الأصناف والمثيل

لقد تعلمت سابقا (انظر للصفحة 68) أن المتغيرات التي يتم تعريفها داخل دالة هي متغيرات خاصة (محلية)، لا يمكنها الوصول إلى تعليمات الموجودة خارج الدالة. وهذا يتيح لها استخدام نفس الأسماء في أجزاء مختلفة من البرنامج، من دون خطر تداخلهم.

لوصف هذا بطريقة أخرى، يمكن أن نقول أن كل دالة لديها مساحة أسمائها، بغض النظر عن مساحة الأسماء الرئيسة.

و تعلمت أيضا أن التعليمات الموجودة داخل الدالة يمكنها الوصول إلى المتغيرات التي تم تعريفها على مستوى الرئيسي لكن فقط بالتشاور : يمكنها استخدام قيم هذه المتغيرات، لكن ليس تعديلها (إلا لو استدعيت عبارة **global**).

لذا يوجد تسلسل هرمي بين مساحات الأسماء. سوف نرى نفس الشيء عن الأصناف والكائنات. وفي الحقيقة :

- كل صنف لديه مساحة الأسماء الخاصة به. المتغيرات التي هي جزء من الصنف تسمى بمتغيرات الصنف أو سمات الصنف.
- كل كائن مثيل (تم صنعه من صنف) يحصل على مساحة الأسماء الخاصة به. المتغيرات التي هي جزء منها تسمى متغيرات المثيل أو سمات المثيل.
- الأصناف يمكنها استخدام (لكن ليس تعديل) المتغيرات التي تم تعريفها في المستوى الرئيس.
- المثيل يمكنه استخدام (لكن ليس تعديل) متغيرات التي تم تعريفها في المستوى الرئيس للصنف والمتغيرات التي تم تعريفها في المتسوى الرئيس للبرنامج .

على سبيل المثال انظر إلى الصنف **Time()** الذي تم تعريفه سابقا. في الصفحة 190، مثلنا 3 كائنات من هذا الصنف: **rentree**، **recreation** و **rendezVous**. كل واحد تم تمثيله مع قيم مختلفة ومستقلة. يمكننا تعديل وإعادة عرض هذه المتغيرات كل واحدة من هذه الكائنات، دون أن يؤثر أي واحد على الأخر :

```
>>> recreation.heure = 12
>>> rentree.affiche_heure()
10:30:0
>>> recreation.affiche_heure()
12:15:18
```

أرجو الآن أن تقوم بترميز وتجربة المثال أدناه :

```
>>> class Espaces(object):                          # 1
...     aa = 33                                     # 2
...     def affiche(self):                          # 3
...         print(aa, Espaces.aa, self.aa)          # 4
...
>>> aa = 12                                         # 5
>>> essai = Espaces()                               # 6
>>> essai.aa = 67                                   # 7
>>> essai.affiche()                                 # 8
12 33 67
>>> print(aa, Espaces.aa, essai.aa)                 # 9
12 33 67
```

في هذا المثال، الاسم **aa** يستخدم لتعريف ثلاثة متغيرات مختلفة : الأولى في مساحة أسماء الصنف (في السطر الثاني)، وواحدة أخرى في مساحة الأسماء الرئيسة (في السطر الخامس)، وأخيرا في مساحة أسماء المثيل (في السطر السابع).

في السطر الرابع والسطر التاسع يبين كيفية الوصول إلى هذه الفضائات الثلاثة للأسماء (في داخل الصف، أو في المستوى الرئيسي)، وذلك باستخدام تأهيل بالنقاط. ولاحظ أيضا مرة أخرى استخدام **self** للإشارة إلى المثيل في داخل تعريف الصنف .

الإرث

الأصناف هي الأداة الرئيسية للبرمجة الشيئية (أو OOP)، التي تعتبر اليوم تقنية البرمجة الأكثر فعالية. واحدة من المزايا الرئيسية لهذا النوع من البرمجة يكمن في أن نتمكن من استخدام دائما فئة موجود لإنشاء واحدة أخرى، والتي سوف ترث جميع خصائصه لكن تستطيع تغيير بعضها أو إضافة خصائص جديدة. وهذه العملية تدعى الاشتقاق. ويقوم بإنشاء تسلسل هرمي للأصناف من العامة إلى الخاصة.

على سبيل المثال سوف نعرف الصنف **()Mammifere** الذي يحتوي على خصائص هذه المجموعة من الحيوانات. ومن هذا الصنف الأصل (الأم)، يمكننا اشتقاق واحدة أو أكثر من هذه الأصناف الفرعية مثل صنف **()Primate**، وصنف **()Rongeur**، وصنف **()Carnivore**، إلخ.، والتي سوف ترث جميع خصائص الصنف **()Mammifere**، ثم نضيف إليها خصائص هذه المجموعة.

بداية من الصنف **()Carnivore**، نستطيع اشتقاق صنف **()Belette**، والصنف **()Loup**، والصنف **()Chien**، إلخ. وهم يرثون جميع خصائص صنف الأصل والصنف الذي فوقه. مثال :

```
>>> class Mammifere(object):
...     caract1 = "il allaite ses petits ;"

>>> class Carnivore(Mammifere):
...     caract2 = "il se nourrit de la chair de ses proies ;"

>>> class Chien(Carnivore):
...     caract3 = "son cri s'appelle aboiement ;"

>>> mirza = Chien()
>>> print(mirza.caract1, mirza.caract2, mirza.caract3)
il allaite ses petits ; il se nourrit de la chair de ses proies ;
son cri s'appelle aboiement ;
```

في هذا المثال، نرى أن الكائن **mirza** ، لديه مثيل لصنف **()Chien**، ولم يرث فقط سمات التي تم تعريفها في هذا الصنف، بل حتى سمات التي تم تعريفها للأصناف الأصل.

لقد رأينا في هذا المثال كيف نشرع في اشتقاق صنف من الصنف الأصل (الأم) : باستخدام تعليمة **class**، ثم كالعادة اسم الذي نريد تعيينه لهذا الصنف، وفي ما بين قوسين اسم الصنف الأصل. الأصناف الأول التي تستمد منها الأصناف الأخرى تسمى الأسلاف.

لاحظ أن السمات المستخدمة في هذا المثال هي سمات أصناف (و ليست سمات المثيل). المثيل **mirza** يمكنه الوصول لهذه السمات، لكن ليس تغييرها:

```
>>> mirza.caract2 = "son corps est couvert de poils ;"    # 1
>>> print(mirza.caract2)                                   # 2
son corps est couvert de poils ;                           # 3
>>> fido = Chien()                                         # 4
>>> print(fido.caract2)                                    # 5
il se nourrit de la chair de ses proies ;                  # 6
```

في المثال الجديد، في السطر الأول، لم يغير سمة **caract2** للصنف **Carnivore()**. على عكس ما تراه في السطر الثالث. يمكنك التحقق من خلال صنع مثيل جديد **fido** (من السطر 4 إلى السطر 6).

إذا كنت قد فهمت جيدا الفقرات السابقة، فإنك سوف تفهم أن التعليمة في السطر 1 تصنع متغيرا جديدا لمثيل مرتبط فقط مع الكائن **mirza**. ولذلك يوجد الآن متغيران لهما نفس الاسم **caract2** : واحدة في مساحة أسماء للكائن **mirza**، والثانية في مساحة أسماء الصنف **Carnivore()**.

لكن كيف يمكننا أن نفسر ما يحدث في السطرين 2 و 3 ؟

كما رأينا أعلاه، المثيل **mirza** يمكنه الوصول للمتغيرات التي تقع في مساحة أسماء الأصناف الأصل. فإذا وجد متغيرات بنفس الاسم في العديد من هذه المساحات، فأي واحدة سيختار أثناء تشغيل التعليمة مثل التي في السطر الثاني ؟

لحل هذا التعارض، يتبع بيثون قاعدة الأولويات في غاية البساطة. فعندما تسأله لاستخدام قيمة متغير يدعى **alpha**، على سبيل المثال، فإنه يبدأ في البحث عن هذا الاسم في المساحة المحلية (الداخلية). فإذا وجد المتغير **alpha** في المساحة المحلية، يستخدمه ويوقف البحث. وإذا لم يجده، يقوم بيثون بفحص مساحة أسماء هيكل الأصل، ثم هيكل فوق الأصل، وإلخ إلى أن يصل إلى المستوى الرئيسي للبرنامج.

في السطر الثاني من مثالنا، فإنه متغير المثيل هو الذي يتم استخدامه. وفي السطر الخامس، فإنه في المستوى الصنف فوق الأصل يوجد به متغير باسم **caract2** لذا هذا الذي قام بعرضه ..

الميراث والتعدد

حلل بعناية السكريبت في الصفحة التالية. أنها تنفذ عدة مفاهيم مذكورة أعلاه، ولا سيما مفهوم الميراث.

لفهم السكريبت، يجب عليك تذكر بعض المفاهيم الكيميائية. في مادة الكيمياء الخاصة بك، فإنك بالتأكيد قد تعلمت أن الذرات هي كيانات تتكون من عدد من البروتونات(جسيمات مشحونة بطاقة كهربائية موجبة)، والإلكترونات (شحنتها سالبة) والنيوترونات (محايدة).

نوع الذرة (أو العنصر) يتم تحديده من خلال عدد البروتونات، والذي يسمى أيضا بالعدد الذري. في حالته الأساسية، الذرة تحتوي على إلكترونات بنفس عدد البروتونات، وبالتالي فهي متعادلة كهربائيا. كما أن لها عدد متغير من النيوترونات، لكن هذا لا يؤثر بأي شكل من الأشكال على الشحنة الكهربائية عموما.

في ظروف معينة، يمكن للذرة أن تكتسب أو أن تفقد إلكترون. وفي هذه الحالة يكتسب شحنة كهربائية عامة ويصبح أيون (الأيون السلبي إذا إكتسبت الذرة إلكترون أو أكثر والأيون الإيجابي إذا فقدت). الشحنة الكهربائية للأيون تساوي الفرق بين عدد البروتونات وعدد الإلكترونات التي تحتويها.

السكريبت في الصفحة التالية يصنع كائنات **()Atome** وكائنات **()Ion**. ذكرنا بالأعلى أن الأيون هو ببساطة ذرة تم تغييرها. في برمجتنا، الصنف الذي يقوم بتعريف كائنات **()Ion** سيكون صنف فرعي من الصنف **()Atome** : سوف يرث جميع سماته وأساليبه، ثم يضيف عليها خصائصه.

واحدة من هذه الأساليب التي تمت إضافتها (الأسلوب **()affiche**) تستبدل الأسلوب من نفس اسم الموروث من الصنف **()Atome**. الأصناف **()Atome** و **()Ion** لديهم كل واحد منهم أسلوب بنفس الاسم، لكن يعمل بشكل مختلف. لنتحدث عن هذه الحالة من تعدد الأشكال. يمكننا القول أن الأسلوب **()affiche** من الصنف **()Atome** كان فوق الطاقة .

من الواضح أنه من الممكن صنع مثيل لأي عدد من الذرات والأيونات من هذين الصنفين. أو واحدة داخل الاخرى، الصنف **()Atome**ـ يجب عليه أن يحتوي على نسخة مبسطة من الجدول الدوري للعناصر (الجدول الدوري)، بحيث يمكنك تعيين اسم العنصر الكيميائي، وعدد النيوترونات، لكل كائن صنعته. كما أنه ليس من المرغوب نسخ هذا الجدول لكل مثيل، سوف نضعه في سمة الصنف. وبالتالي هذ الجدول لن يتواجد إلا في مكان واحد في الذاكرة، حتى تبقى في متناول جميع الكائنات التي سيتم إنتاجها من هذا الصنف .

```
class Atome:
    """atomes simplifiés, choisis parmi les 10 premiers éléments du TP"""
    table =[None, ('hydrogène',0),('hélium',2),('lithium',4),
            ('béryllium',5),('bore',6),('carbone',6),('azote',7),
            ('oxygène',8),('fluor',10),('néon',10)]

    def __init__(self, nat):
        "le n° atomique détermine le n. de protons, d'électrons et de neutrons"
        self.np, self.ne = nat, nat          # nat = العدد الذري
        self.nn = Atome.table[nat][1]

    def affiche(self):
        print()
        print("Nom de l'élément :", Atome.table[self.np][0])
        print("{0} protons, {1} électrons, {2} neutrons".\
              format(self.np, self.ne, self.nn))
```

```
class Ion(Atome):
    """les ions sont des atomes qui ont gagné ou perdu des électrons"""

    def __init__(self, nat, charge):
        "le n° atomique et la charge électrique déterminent l'ion"
        Atome.__init__(self, nat)
        self.ne = self.ne - charge
        self.charge = charge

    def affiche(self):
        Atome.affiche(self)
        print("Particule électrisée. Charge =", self.charge)

### البرنامج الرئيسي : ###

a1 = Atome(5)
a2 = Ion(3, 1)
a3 = Ion(8, -2)
a1.affiche()
a2.affiche()
a3.affiche()
```

عند تشغيل هذا البرنامج سوف يظهر التالي :

```
Nom de l'élément : bore
5 protons, 5 électrons, 6 neutrons

Nom de l'élément : lithium
3 protons, 2 électrons, 4 neutrons
Particule électrisée. Charge = 1

Nom de l'élément : oxygène
8 protons, 10 électrons, 8 neutrons
Particule électrisée. Charge = -2
```

في مستوى البرنامج الرئيسي، يمكنك أن ترى أننا مثلنا كائنات **Atome()** عددها الذري (الذي يجب أن يكون ما بين 1 و 10). لتمثيل كائنات **Ion()**ـ يجب أن نوفر العدد الذري وشحنته الكهربائية العامة (موجبة أو سالبة). نفس الأسلوب **affiche()** يبين خصائص هذه الكائنات، سواء أن كانت ذرات أو أيونات، وفي حالة الأيون خط إضافي (تعدد الأشكال) .

التعليقات

تعريف الصنف **Atome()** يبدأ بتعيين المتغير **table**. المتغير الذي يتم تعريفه هنا هو جزء من مساحة أسماء الصنف. إذا هذا هو سمة الصنف، الذي نضع فيها قائمة المعلومات حول 10 أول عناصر الجدول الدوري لمنديلف (اسم الشخص الذي اخترع هذا الجدول).

لكل واحدة من هذه العناصر، قائمة تحتوي على مصفوفة مغلقة (tuple) : (اسم العنصر عدد النيوترونات)، والمؤشر الذي يتوافق مع العدد الذري. وبما أنه لا توجد عناصر عددها الذري صفر، لذا وضعنا للمؤشر صفر في القائمة، الكائن الخاص. يمكننا وضع هناك أي قيمة أخرى، لأن هذا المؤشر لن يستخدم الكائن None لبيثون .

تليها تعارف الأسلوبين :

- المنشئ **__init__()** يستخدم أساسا هنا لصنع ثلاثة سمات المثيل، لتخزين في الذاكرة أعداد البروتونات والإلكترونات والنيوترونات على التوالي لكل كائن ذرة صنع من هذا الصنف (تذكر أن سمات المثيل هي متغيرات مرتبطة برامتر **self**).

  لاحظ أن التقنية المستخدمة للحصول على عدد النيوترونات من سمة الصنف، مما يدل على اسم الصنف مؤهل بنقاط، مثل في التعليمة: `self.nn = Atome.table[nat][1]`.
- الأسلوب **affiche()** يستخدم سمات المثيل، لإيجاد عدد البروتونات والألكترونات والنيوترونات للكائن الحالي، وسمة الصنف (التي هي مشتركة بين جميع الكائنات) لاستخراج اسم العنصر المطابق .

تعريف صنف **Ion()** يتضمن في أقواسه اسم الصنف **Atome()** أعلاه.

أساليب هذا الصنف هي بدائل الصنف **Atome()**. سيقومون باحتمال استدعاء هذه. هذه الملاحظة مهمة : كيف يمكننا، في إطار تعريف الصنف، استدعاء الأسلوب الذي تم تعريفه في صنف آخر.

لا ينبغي أن نغفل، في الحقيقة، عن الأسلوب الذي يرتبط دائما بالمثيل الذي سيتم صنعه من هذا الصنف (المثيل يمثل بواسطة **self** في تعريفه). إذا كان الأسلوب يجب عليه استدعاء أسلوب آخر تم تعريفه في صنف آخر، يجب أن يكون قادر على نقل المرجع المثيل الذي ينبغي أن يقترن بها. كيف نفعل هذا؟ هذا بسيط جدا:

> في تعريف الصنف، قد نرغب باستدعاء أسلوب تم تعريفه في صنف اخر، يمكننا ببساطة استدعاؤها مباشرة، عن طريق صنف اخر، بتمرير مرجع المثيل كبرامتر أول .

وبالتالي في سكريبتنا، على سبيل المثال، الأسلوب للصنف **affiche()** للصنف **Ion()** يمكنه استدعاء الأسلوب **affiche()** للصنف **Atome()** : المعلومات المعروضة ستكون الكائن-الأيون الحالي، لأن مرجعه تم تمريره في تعليمة تدعى:

```
        Atome.affiche(self)
```

في هذه التعليمة، **self** بالطبع هو مرجع المثيل الحالي. بنفس الطريقة (سوف نرى أمثلة أخرى كثيرة فيما بعد)، الأسلوب المنشئ للصنف **Ion**() استدعى الأسلوب المنشئ للصنفه الأصل، في :

```
        Atome.__init__(self, nat)
```

هذا الاستدعاء ضروري، بحيث يتم تهيئة كائنات الصنف **Ion**() بنفس الطريقة كائنات الصنف Si nous .**Atome()** إذا كنا لم نقم بهذا الاستدعاء، الكائنات-الأيونات لن يرثوا تلقائيا سمات **np**، **ne** و **nn**، لأن هذه سمات المثيل تم صنعها بواسطة أسلوب المنشئ للصنف **Atome**()، وذلك لن يتم استدعاؤه تلقائيا عند إنشاء كائنات من صنف مشتق.

افهم إذًا أن الميراث لا ينطبق إلا على الأصناف، وليس على مثيل هذه الأصناف. وعندما نقول أن صنفا مشتقا يرث جميع خصائص صنفه الأصل، هذا لا يعني أن الخصائص المثيل للصنف الأصل ينقل تلقائيا لأمثال الصنف الإبن. ووفقا لذلك، تذكر:

> في أسلوب منشئ الصنف المشتق، يجب تقريبا دائما أن تقوم باستدعاء أسلوب المنشئ من صنف الأصل .

Résumé : définition et utilisation d'une classe

```
####################################
# Programme Python type            #
# auteur : G.Swinnen, Liège, 2009  #
# licence : GPL                    #
####################################

class Point(object):
    """point géométrique"""
    def __init__(self, x, y):
        self.x = x
        self.y = y

class Rectangle(object):
    """rectangle"""
    def __init__(self, ang, lar, hau):
        self.ang = ang
        self.lar = lar
        self.hau = hau

    def trouveCentre(self):
        xc = self.ang.x + self.lar /2
        yc = self.ang.y + self.hau /2
        return Point(xc, yc)

class Carre(Rectangle):
    """carré = rectangle particulier"""
    def __init__(self, coin, cote):
        Rectangle.__init__(self,
                coin, cote, cote)
        self.cote = cote

    def surface(self):
        return self.cote**2

###########################
## Programme principal : ##

# coord. de 2 coins sup. gauches :
csgR = Point(40,30)
csgC = Point(10,25)

# "boîtes" rectangulaire et carrée :
boiteR = Rectangle(csgR, 100, 50)
boiteC = Carre(csgC, 40)

# Coordonnées du centre pour chacune :
cR = boiteR.trouveCentre()
cC = boiteC.trouveCentre()

print("centre du rect. :", cR.x, cR.y)
print("centre du carré :", cC.x, cC.y)

print("surf. du carré :", end=' ')
print(boiteC.surface())
```

La classe est un moule servant à produire des objets. Chacun d'eux sera une instance de la classe considérée.

Les instances de la classe Point() seront des objets très simples qui possèderont seulement un attribut 'x' et un attribut 'y' ; ils ne seront dotés d'aucune méthode.

Le paramètre SELF désigne toutes les instances qui seront produites à partir de cette classe.

Les instances de la classe Rectangle() posséderont trois attributs. Le premier ('ang') doit être lui-même un objet de classe Point(). Il servira à mémoriser les coordonnées de l'angle supérieur gauche du rectangle. Les deux autres contiendront sa largeur et sa hauteur.

La classe Rectangle() comporte une méthode, qui renverra un objet de classe Point() au programme appelant.

Carre() est une classe dérivée, qui hérite les attributs et méthodes de la classe Rectangle().
Son constructeur doit faire appel au constructeur de la classe parente, en lui transmettant la référence de l'instance en cours de création (self) comme premier argument.

La classe Carre() comporte une méthode de plus que sa classe parente.

Pour créer (ou instancier) un objet, il suffit d'appeler une classe comme on appelle une fonction. La valeur renvoyée est une nouvelle instance de cette classe. Les instructions ci-contre créent donc deux objets de la classe Point() ...

... et celles-ci, encore deux autres objets.

Note : par convention, on donne aux classes des noms commençant par une majuscule.

La méthode trouveCentre() fonctionne pour les objets des deux types, puisque la classe Carre() a hérité de la classe Rectangle().

La méthode surface(), par contre, ne peut être invoquée que pour les objets Carre().

وحدات تحتوي على مكتبات الأصناف

أنت تعرف بالفعل منذ وقت طويل استخدام وحدات بيثون (انظر للصفحات 52 و79). وأنت تعرف أنها يتم تجميعها في مكتبات متكونة من الأصناف والدالات. كتمرين مرجعة، سوف تقوم بصنع وحدة جديدة من الأصناف، بترميز أسطر التعليمات في الأسفل في ملف وحدة التي سوف تسميها **formes.py** :

```
class Rectangle(object):
    "Classe de rectangles"
    def __init__(self, longueur =0, largeur =0):
        self.L = longueur
        self.l = largeur
        self.nom ="rectangle"

    def perimetre(self):
        return "({0:d} + {1:d}) * 2 = {2:d}".\
                format(self.L, self.l, (self.L + self.l)*2)
    def surface(self):
        return "{0:d} * {1:d} = {2:d}".format(self.L, self.l, self.L*self.l)

    def mesures(self):
        print("Un {0} de {1:d} sur {2:d}".format(self.nom, self.L, self.l))
        print("a une surface de {0}".format(self.surface()))
        print("et un périmètre de {0}\n".format(self.perimetre()))

class Carre(Rectangle):
    "Classe de carrés"
    def __init__(self, cote):
        Rectangle.__init__(self, cote, cote)
        self.nom ="carré"

if __name__ == "__main__":
    r1 = Rectangle(15, 30)
    r1.mesures()
    c1 = Carre(13)
    c1.mesures()
```

عند حفظ هذه الوحدة، يمكنك استخدامه بطريقتين : إما أن تقوم بتشغيله مثل أي برنامج، وإما من خلال استدعائه في أي سكريبت أو من سطر الأوامر، لاستخدام أصنافه. انظر لهذا المثال :

```
>>> import formes
>>> f1 = formes.Rectangle(27, 12)
>>> f1.mesures()
Un rectangle de 27 sur 12
a une surface de 27 * 12 = 324
et un périmètre de (27 + 12) * 2 = 78

>>> f2 = formes.Carre(13)
>>> f2.mesures()
Un carré de 13 sur 13
a une surface de 13 * 13 = 169
et un périmètre de (13 + 13) * 2 = 52
```

نحن نرى في هذا السكريبت أن الصنف **Carre()** تم إنشاؤه من الصنف **Rectangle()** لذا هو يرث جميع خصائصه. وبعبارة أخرى، الصنف **Carre()** هو ابن الصنف **Rectangle()**.

قد تلاحظ مرة أخرى أن منشئ الصنف **Carre()** يجب عليه استدعاء منشئ الصنف الأصل (`Rectangle.__init__(self`، ...))، وأن يمرر له مرجع المثيل (**self**) كأول برامتر.

أما بالنسبة للتعليمة :

```
if __name__ == "__main__":
```

تم وضعها في نهاية الوحدة، وتستخدم لمعرفة إذا كانت الوحدة تم تشغيلها كبرنامج مستقل (في هذه الحالة يجب أن يتم تنفيذ التعليمات التي تليها)، أو تم استخدامه كمكتبة لصنف تم استدعاؤه في مكان آخر. في هذه الحالة هذا الجزء من الكود ليس له أي تأثير .

تمارين

5.12 عرف الصنف **Cercle()**. والأصناف التي تم صنعا من هذا الصنف تكون دوائر بأحجام مختلفة. بالإضافة إلى أسلوب منشئ (الذي سوف يستخدم البرامتر **rayon**)، يمكنك تعريف الأسلوب **surface()** الذي يقوم بإرجاع مساحة الدائرة.

قم بتعريف الصنف **Cylindre()** المشتق من الذي قبله. المنشئ لهذا الصنف الجديد يتضمن برامترين **rayon** و **hauteur**. أضف الأسلوب **volume()** الذي سيقوم بإرجاع حجم الأسطوانة (تذكير : حجم الإسطوانة = مساحة المقطع × الإرتفاع) .

مثال استخدام لهذا الصنف :

```
>>> cyl = Cylindre(5, 7)
>>> print(cyl.surface())
78.54
>>> print(cyl.volume())
549.78
```

6.12 أكمل التمرين السابق بإضافة صنف جديد **Cone()** سوف يشتق هذه المرة من الصنف **Cylindre()**، والمنشئ يتضمن أضا برامترين **rayon** و **hauteur**. هذا الصنف الجديد لديه أسلوبه **volume()**، والذي سيقوم بإرجاع حجم المخروط (تذكر حجم المخروط = حجم الإسطوانة المقابلة مقسومة على 3).

مثال لاستخدام لهذا الصنف :

```
>>> co = Cone(5,7)
>>> print(co.volume())
183.26
```

7.12 عرف الصنف **JeuDeCartes()** الذي يسمح بتمثيل الكائنات التي سلوكها مشابه لسلوك لعبة ورق حقيقية. الصنف يجب عليه أن يحتوي على الأقل على أربعة الأساليب التالية :

• أسلوب المنشئ : إنشاء وملء قائمة ثم 52 عنصر والتي كل واحد منها مكون من مصفوفة مغلقة (tuple) من عددين صحيحين. هذه قائمة المصفوفة المغلقة (tuple) تحتوي على خصائص لكل واحدة من 52 بطاقة. لكل واحدة منها، يجب تخزين بشكل منفصل عدد صحيح يشير إلى قيمة (2، 3، 4، 5، 6، 7، 8، 9، 10، 11، 12، 13، 14، الأربعة الأخيرة قيم جاك والملكة والملك والأس)، والعدد الصحيح الأخر يشير إلى لون الورقة (و هذا معناه 0،1،2،3 للقلب والماس وو النوادي وبستوني).

في هذه القائمة، العنصر (11،2) معناه جاك النوادي، والقائمة يجب أن تكتمل بنوع :

[(14,3) ,(13,3) ,(12,3) ..... ,(4,0) ,(3,0) ,(3,0) ,(0 ,2)]

• الأسلوب **nom_carte()** : هذا الأسلوب يجب عليه أن يقوم بإرجاع في شكل سلسلة، بها هوية البطاقة ويجب أن تكون مصفوفة مغلقة (tuple) لوصف البرامتر. على سبيل الامثال، التعليمة: `print(jeu.nom_carte((14, 3)))` يجب أن تعرض: `As de pique`

• الأسلوب **battre()** : كما يعلم الجميع، معناه خلط الأوراق. هذا الأسلوب سوف يخلط قائمة العناصر التي تحتوي على الأوراق، بغض النظر عن العدد .

• الأسلوب **tirer()** : عندما يتم استدعاء هذا الأسلوب، يتم سحب ورقة . المصفوفة المغلقة (tuple) الذي يحتوي على القيمة واللون يتم إرساله للبرنامج الذي استدعاه. يتم دائما سحب (إزالة من القائمة) أول ورقة في القائمة. فإذا تم استدعاء هذا الأسلوب ولم يعد يوجد أوراق في القائمة، يجب إرسال الكائن الخاص **None** إلى البرنامج الذي استدعاه.

أمثلة استخدام للصنف **JeuDeCartes()** :

```
jeu = JeuDeCartes()              # تمثيل كائن
jeu.battre()                     # خلط الأوراق
for n in range(53):              # صنع 52 ورقة:
   c = jeu.tirer()
   if c == None:                 # لا توجد أي ورقة في القائمة
      print('Terminé !')
   else:
      print(jeu.nom_carte(c))    # قيمة ولون الورقة
```

8.12 أكمل التمرين السابق : عرف لاعبين A و B. قم بتمثيل ورقتي لعب (واحد لكل لاعب) وقم بخلطها. ثم بمساعدة حلقة التكرار، اسحب 52 مرة ورقة لكل واحد من هذان اللاعبان وقارن قيمتهما. إذا كانت الأول الأكبر قيمة يتم إضافة نقطة للاعب A. فإذا حدث العكس فيتم إضافة نقطة للاعب B. فإذا كانت القيمتان متعادلتين، يتم المرور إلى السحب التالي. عند انتهاء الحلقة، أحسب نقاط A و B لمعرفة الفائز .

9.12 اكتب سكريبت جديد يقوم باسترداد كود التمرين 12.2 (الحساب المصرفي) عن طريق استدعائه كوحدة. وعرف صنف **()CompteEpargne**ـ مشتقة من صنف **()CompteBancaire** التي تم استدعاؤها، والتي تسمح بصنع حسابات مدخرات يضاف إليه بعض الفائدة بعد مرور الوقت. للتبسيط، نحن نفترض أنه يتم حساب فائدة شهرية.

منشئ صنفك الجديد يجب عليه أن يهيئ فائدة شهرية بالتقصير تساوي 0,3 %. وأسلوب **changeTaux(valeur)** يسمح بتغيير معدل الفائدة .

الأسلوب **capitalisation(nombreMois)** يجب أن:

- يعرض عدد الأشهر والفائدة التي يتم أخذها في الحساب .
- يحسب المال المتحصل عليه من خلال الفائدة والأشهر التي تم اختيارها .

أمثلة استخدام هذا الصنف :

```
>>> c1 = CompteEpargne('Duvivier', 600)
>>> c1.depot(350)
>>> c1.affiche()
Le solde du compte bancaire de Duvivier est de 950 euros.
>>> c1.capitalisation(12)
Capitalisation sur 12 mois au taux mensuel de 0.3 %.
>>> c1.affiche()
Le solde du compte bancaire de Duvivier est de 984.769981274 euros.
>>> c1.changeTaux(.5)
>>> c1.capitalisation(12)
Capitalisation sur 12 mois au taux mensuel de 0.5 %.
>>> c1.affiche()
Le solde du compte bancaire de Duvivier est de 1045.50843891 euros.
```

13

الأصناف وواجهات المستخدم الرسومية

البرمجة الشيئية هي مناسبة خاصة لتطوير التطبيقات مع واجهات المستخدم الرسومية. مكتبات الأصناف مثل *tkinter* أو *wxPython* توفر أساس ودجات واسعا جدا، حيث نستطيع أن نكيف إحتياجاتنا من الاشتقاق. في هذا الفصل، سوف نستخدم مرة أخرى مكتبة *tkinter*، لكن سوف نطبق المفاهيم الموضحة في الصفحات السابقة، وسنسعى لتسليط الضوء على مزايا البرمجة الشيئية في برامجنا .

كود الألوان : مشروع صغير مغلف بشكل جيد

سنبدأ مع مشروع صغير مستوحى من الدورات في مجال الألكترونيات. التطبيق الذي سنصفه أدناه يمكنه العثور بسرعة على رمز 3 ألوان المطابقة للمقاوم الكهربائي لقيمة محددة جدا.

للتذكير، إن وظيفة المقاوم هي مقاومة (اعتراض) قليل أو كثير من تدفق التيار الكهربائي. المقاوم يظهر بشكل ملموس في شكل أنبوبي من أجزاء صغيرة بها ثلاثة خطوط من الألوان (عادة 3). هذه الخطوط تشير إلى القيمة العددية للمقاومة، اعتمادا على التالي :

أسود = 0 ; بني = 1 ; أحمر = 2 ; برتقالي = 3 ; أصفر = 4 ;
أخضر = 5 ; أزرق = 6 ; بنفسجي = 7 ; رمادي = 8 ; أبيض = 9.

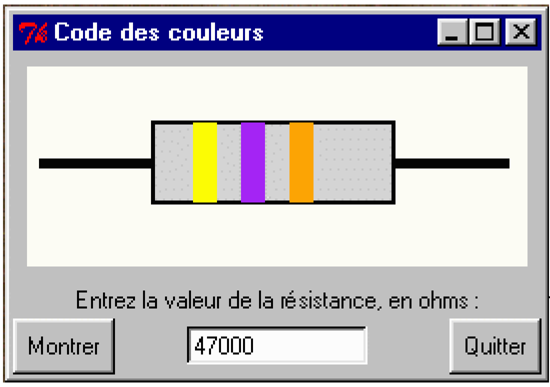
المقاومة موجهة بحيث يتم وضع الخطوط (الشرائح) الملونة على اليسار. قيمة المقاومة - تعرف بالأوم (**Ω**) – يتم قراءة الخطوط (الأشرطة) من اليسار : أول خطين يشيران إلى أول رقمين من القيمة الرقمية. ثم يجب إلحاق هذين الرقمين بعدد من الأصفار مساوٍ للخط الثالث .

مثال : الألوان من اليسار هي : أصفر وبنفسجي وأخضر.

قيمة المقاوم هي 4700000 **Ω**، أو 4700 **kΩ**، أو 4,7 **MΩ** .

هذا النظام لا يسمح بتوضيح القيمة الرقمية إلا مع رقمين فقط. ومع ذلك هذا منتشر على نطاق واسع. وهو كافٍ بالنسبة لمعظم التطبيقات "العادية" (الراديو، التلفاز، إلخ) .

مواصفات برنامجنا



يجب على برنامجنا أن يقوم بعرض نافذة تحتوي على رسم للمقاوم، ويجب على المستخدم إدخال القيمة العددية للمقاوم ثم الضغط على <Montrer> وهذا سيتسبب في تغيير رسم المقاوم، بطريقة لتتوافق ألوان الخطوط مع القيمة التي أدخلها المستخدم.

عائق : يجب على البرنامج قبول أي قيمة رقمية (عدد صحيح أو حقيقي) وذلك في حدود من 10 إلى 10^{11} **Ω**. على سبيل المثال، القيمة 4.78e6 يجب أن تكون مقبولة ويجب تحويلها إلى 4800000 **Ω**.

تطبيق ملموس

سوف نبني جسم هذا التطبيق البسيط على شكل صنف. نحن نريد أن نظهر لك كيف أن الصنف يمكنه خدمة مساحة الأسماء المشتركة، حيث يمكنك تغليف متغيراتنا ودالاتنا. والميزة الرئيسية هو أن يسمح لك بتمرير متغيرات عامة. في الواقع :

- بدء تشغيل التطبيق لتلخيص مثيل كائن هذا الصنف.
- الدالات التي نريد تنفيذها ستكون أساليب لهذا الكائن-التطبيق.
- و ضمن هذه الأساليب، يمكنك ببساطة إرفاق اسم المتغير للبرامتر **self** لكي يمكنك الوصول لهذا المتغير من أي مكان من داخل الكائن. المتغير المثيل يشبه تماما متغير عمومي (لكن فقط داخل الكائن)، لأن جميع الأساليب للكائن يمكنهم الوصول إلى **self**.

```
class Application(object):
    def __init__(self):
        """Constructeur de la fenêtre principale"""
        self.root =Tk()
        self.root.title('Code des couleurs')
        self.dessineResistance()
        Label(self.root, text ="Entrez la valeur de la résistance, en ohms :").\
                grid(row =2, column =1, columnspan =3)
        Button(self.root, text ='Montrer', command =self.changeCouleurs).\
                grid(row =3, column =1)
        Button(self.root, text ='Quitter', command =self.root.quit).\
                grid(row =3, column =3)
        self.entree = Entry(self.root, width =14)
```

```
        self.entree.grid(row =3, column =2)
        # الألوان للقيمة من 0 إلى 9 :
        self.cc = ['black','brown','red','orange','yellow',
                   'green','blue','purple','grey','white']
        self.root.mainloop()

    def dessineResistance(self):
        """Canevas avec un modèle de résistance à trois lignes colorées"""
        self.can = Canvas(self.root, width=250, height =100, bg ='ivory')
        self.can.grid(row =1, column =1, columnspan =3, pady =5, padx =5)
        self.can.create_line(10, 50, 240, 50, width =5)         # fils
        self.can.create_rectangle(65, 30, 185, 70, fill ='light grey', width =2)
        # رسم ثلاثة خطوط ملونة (أسود في البداية) :
        self.ligne =[]            # تخزين الخطوط الثلاثة في قائمة واحدة
        for x in range(85,150,24):
            self.ligne.append(self.can.create_rectangle(x,30,x+12,70,
                                                        fill='black',width=0))

    def changeCouleurs(self):
        """Affichage des couleurs correspondant à la valeur entrée"""
        self.v1ch = self.entree.get()
        # هذا الأسلوب يقوم بإرجاع سلسلة واحدة
        try:
            v = float(self.v1ch)          # تحويلها إلى قيمة رقيمة
        except:
            err =1                        # خطأ: المعطيات ليست رقمية
        else:
            err =0
        if err ==1 or v < 10 or v > 1e11 :
            self.signaleErreur()          # المدخلات خاطئة أو خارجة
        else:
            li =[0]*3                     # قائمة من 3 رموز لعرض
            logv = int(log10(v))          # الجزء الصحيح من الخوارزمية
            ordgr = 10**logv              # ترتيب الحجم
            # استخراج أول أرقام هامة :
            li[0] = int(v/ordgr)          # جزء صحيح
            decim = v/ordgr - li[0]       # جزء حقيقي
            # استخراج أول أرقام هامة :
            li[1] = int(decim*10 +.5)     # +.5 لتقريب بشكل صحيح
            # عدد الأصفار المرتبطة بالرقمين الكبيرين:
            li[2] = logv -1
            # تلوين الخطوط الثلاثة:
            for n in range(3):
                self.can.itemconfigure(self.ligne[n], fill =self.cc[li[n]])

    def signaleErreur(self):
        self.entree.configure(bg ='red')          # تلوين خلفية
        self.root.after(1000, self.videEntree)    # امسحهُ بعد 1 ثانية

    def videEntree(self):
        self.entree.configure(bg ='white')        # استعد الخلفي البيضاء
        self.entree.delete(0, len(self.v1ch))     # ازالة الحروف المكتوبة

# البرنامج الرئيسي :
if __name__ == '__main__':
    from tkinter import *
    from math import log10                 # خوارزمية بأساس 10
    f = Application()                      # تمثيل كائن التطبيق
```

تعليقات

- السطر الأول : الصنف تم تعريفه من صنف مستقل (أي أنه لا يستمد من أي صنف أصل، لكن فقط الكائن، وهو سلف جميع الأصناف الأخرى) .
- الأسطر من 2 إلى 14 : منشئ صنف المثيل للويدجات المطلوبة : مساحة رسوم، ملصق (Label) وأزرار. ولتحسين إمكانية قراءة البرنامج، وضعنا مثيل للوحة (مع رسم للمقاوم) في أسلوب منفصل: **dessineResistance()**. يرجى ملاحظة أنه للحصول على كود أصغر حجما، نحن لا نقوم بحفظ الأزرار والملصقات (Label) في متغيرات (كما شرحنا سابقا في الصفحة 104)، وذلك لأننا لا نريد صنع مرجع لها في أماكن مختلفة من البرنامج. نحن استخدمنا لأماكن الويدجات في النافذة الأسلوب **grid()** الذي وصفناه في الصفحة 101.
- الأسطر من 15 إلى 17 : يتم تخزين كود الألوان في قائمة بسيطة.
- السطر 18 : التعليمة الأخيرة لمنشئ بداية البرنامج. فإذا كنت تفضل بدء البرنامج بشكل مستقل عن صنعه، يجب عليك في هذه الحالة حذف هذا السطر، ونستدعي **mainloop()** في المستوى الرئيسي للبرنامج، بإضافة التعليمة: `f.root.mainloop()` في السطر 71.
- الأسطر 20 إلى 30 : رسم المقاوم يتكون من خط وأول شريحة رمادية فاتحة، لجسم المقاوم وابنيه. وثلاثة مستطيلات أخرى كشرائح ملونة وتتغير ألوانه اعتمادا على مدخلات المستخدم. هذه الشرائح تكون لونها أسود في البداية ومرجعهم في قائمة **self.ligne**.
- الأسطر من 32 إلى 53 : هذه الأسطر تحتوي على أساس وظائف البرنامج. مدخلات المستخدم يتم قبولها في شكل سلاسل نصية. في السطر 36، فإننا نحاول تحويل هذه السلسلة إلى قيمة رقمية من نوع حقيقي. فإذا فشل التحويل، يتم تخزين الخطأ. فإذا تحول إلى قيمة رقمية، نتأكد من أنه داخل النطاق المسموح به (من 10 Ω إلى 10^11 Ω). فإذا تم الكشف عن خطأ، فإننا ننبه المستخدم على أن مدخلاته لها خطأ عن طريق تلوين حقل الإدخال بلون أحمر، ويتم إفراغ محتوياته (الأسطر من 55 إلى 61).
- الأسطر 45 و 46 : إن الرياضيات أتت لنجدتنا لاستخراج القيمة الرقمية من ترتيبها الكبير (هذا معناه، من أقرب 10 أس (قوة)). يرجي الرجوع إلى كتاب رياضيات لمزيد من التوضيح عن اللوغاريتمات .
- الأسطر : 47-48 : بمجرد معرفة نظام القيمة الأسية، سيكون من السهل نسبيا إستخراج العدد الذي تم التعامل مع أول منزلتين معتبرتين. فمثلاً، نفترض أن القيمة التي تم إدخالها هي 31587 . اللوغاريتم لهذا الرقم سيكون 4،50088... وبهذا فالرقم المُدخَل (4) و الذي ستكون القيمة الأسية له (10^4). ولإستخراج رقمه الأول المعتبر، يجب قسمته على 10^4 (10000) و الاحتفاظ فقط بالجزء الصحيح من النتيجة (3).

- السطر من 49 إلى 51 : نتيجة عملية القسمة التي قمنا بها في الفقرة السابقة هي كالتالي: سوف نقوم بأخذ الجزء الكسري للعدد في السطر 49 و الذي هو 0،1687 في مثالنا، فإذا ضربناه في عشرة، ستكون هذه النتيجة الجديدة تحمل جزءاً صحيحاً واحداً (والذي هو في مثالنا ثاني منزلة معتبرة وقيمتها واحد).
- يمكننا بسهولة أن نستخرج الرقم الأخير، لكن بما أن هذا الأخير يجب علينا أن نقوم بتقريبه بشكل صحيح. للقيام بذلك، يجب أن نقوم ببساطة بإضافة 0,5 إلى ناتج الضرب في 10 ليتم تقريبه بشكل صحيح، وذلك قبل استخراج القيمة النهائية. وفي مثالنا هذا : إن النتيجة التي نحصل عليها هي 1،687 + 0،5 = 2.187، و بالتالي فإن العدد الصحيح للنتيجة (2) هي القيمة المقربة التي نبحث عنها.
- السطر 53 : عدد الأصفار التي ترتبط برقمين مهمين الموافقة لحساب ترتيب الحجم. يمكنك ببساطة إزالة وحدة في الخوارزمية.
- السطر 56 : لتعيين لون جديد لكائن يمكن رسمه على اللوحة، نستخدم الأسلوب **itemconfigure**(). سوف نستخدم إذا هذا الأسلوب لتبديل خيار التلوين لكل شريحة ملونة، باستخدام الألوان التي تم استخراجها من **self.cc** grâce à aux بمؤشرات الثلاثة **li[3** ، **li[2]**، **li[1]]** التي تحتوي على ثلاثة أرقام.

تمارين

1.13 قم بتعديل السكريبت بالأعلى بحيث يصبح صورة الخلفية أزرق فاتح (light blue)، ويصبح جسم المقاومة لونه بيج (يشبه اللون البني - beige)، والسلك المقاومة يصبح أنحف، وشرائح الألوان التي تدل على القيمة تصبح أكبر .

2.13 عدل السكريبت في الأعلى بحيث تصبح صورة المرسومة أكبر مرتان .

3.13 عد السكريبت أعلاه بحيث يصبح من الممكن إدخال قيمة المقاوم التي ما بين 1 و 10 P .**Ω**ولهذه القيم الشريحة الأولى الملونة يجب أن تبقى سوداء، والشريحتان المتبقيتان يشيران القيم بـ **Ω** والثانية بـ **Ω**.

4.13 عدل السكريبت أعلاه بحيث أن الزر <Montrer> يصبح غير إلزامي. في سكريبتك المعدل، يكفي أن تضغط على <Enter> بعد إدخال قيمة المقاوم، لتنشط الشاشة .

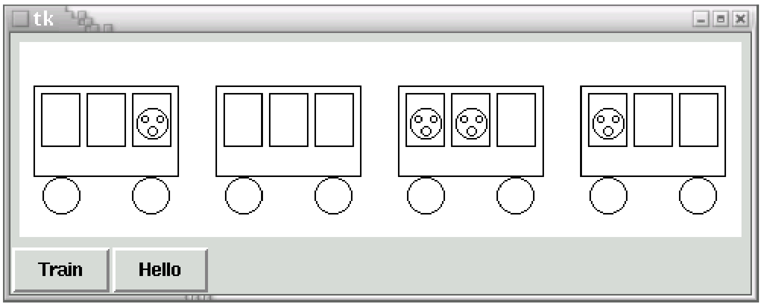
5.13 عدل السكريبت في الأعلى بحيث الشرائح الثلاث الملونة ترجع سوداء في حالة أدخل المستخدم ملاخلات غير مقبولة .

صغير: الميراث. تبادل المعلومات بين الكائنات

في التمرين السابق، نحن لم نستغل سوى ميزة واحدة من الأصناف وهي التغليف. وهذا يسمح لنا بكتابة برامج بمختلف دالاتها (و التي أصبحت أساليب) يمكن كل واحد منها الوصول إلى نفس المجموعة من المتغيرات : جميع التي تم تعريفها كمرفق لـ **self**. ويمكن اعتبار كل هذه المتغيرات كنوع من المتغيرات العامة داخل الكائن.

يجب عليك أن تفهم أنه ليست متغيرات عامة حقيقية. بل هي مقتصرة فقط على داخل الكائن، وينصح بالوصول إليها من الخارج[69]. من ناحية أخرى، فإن جميع الكائنات التي مثلتها من نفس الصنف تمتلك كل واحدة خصائص مجموعة هذه المتغيرات. والتي هي تغليف لهذه الكائنات. وهذا ما يطلق عليه بسمات المثيل.

سوف ننتقل الآن إلى السرعة القصوى، وسننشئ تطبيقا صغيرا على أساس عدة أصناف، لدراسة كيفية تبادل المعلومات بين عدة أصناف من خلال أساليبهم. وسوف نستخدم أيضا هذا التمرين لنبين لك كيف يمكنك تعريف الصنف الأصل لبرنامجك الرسومي باشتقاق من صنف tkinter، وبالتالي الاستفادة من آلية الميراث .



المشروع الذي سنطوره هنا بسيط جدا، لكنه يمكن أن يكون الخطوة الأولى في إنشاء الألعاب : كما سنوفر الأمثلة أدناه (انظر للصفحة 276). ومشروعنا عبارة عن نافذة تحتوي على لوحة وزرين. عندما ينشط الزر الأول، يظهر قطار صغير على اللوحة. وعندما ننشط الزر الثاني، بعض الأشخاص الصغار يظهرون في بعض نوافذ العربات .

المواصفات

البرنامج سيتضمن صنفين :

- الصنف **Application()** سيتم الحصول عليه من خلال اشتقاق أصناف أساسية من tkinter وسوف يقوم بعرض النافذة الرئيسية واللوحة والزرين.
- الصنف المستقل **Wagon()**ـ يسمح بتمثيل في اللوحة 4 كائنات-عربات، ولكل واحدة منها أسلوب **perso()**. وهذا سيتسبب في ظهور الأشخاص الصغار في أي واحدة من النوافذ الثلاث للعربة. البرنامج الرئيس يستدعي هذا الأسلوب بشكل مختلف لمختلف الكائنات-العربات،ثم ليظهر بعض الأشخاص .

---

[69] كما ذكرنا سابقا. يسمح لك بيثون بالوصول إلى سمات المثيل باستخدام تأهيل الأسماء بالنقاط . لغات البرمجة الأخرى تحظر أو تسمح لك فقط من خلال إعلان خاص لهذه السمات (سمات التمييز بين الخاص و العام) .
يرجى ملاحظة أنه في أي حال فمن الغير المستحسن : إن الاستخدام السليم للبرمجة الشيئية تنص على أن تكون قادرا على الوصول إلى سمات الكائنات من خلال طرق محددة (واجهة) .

التطبيق

```
from tkinter import *

def cercle(can, x, y, r):
    "dessin d'un cercle de rayon <r> en <x,y> dans le canevas <can>"
    can.create_oval(x-r, y-r, x+r, y+r)

class Application(Tk):
    def __init__(self):
        Tk.__init__(self)         # منشئ الصنف الأصل
        self.can =Canvas(self, width =475, height =130, bg ="white")
        self.can.pack(side =TOP, padx =5, pady =5)
        Button(self, text ="Train", command =self.dessine).pack(side =LEFT)
        Button(self, text ="Hello", command =self.coucou).pack(side =LEFT)

    def dessine(self):
        "instanciation de 4 wagons dans le canevas"
        self.w1 = Wagon(self.can, 10, 30)
        self.w2 = Wagon(self.can, 130, 30)
        self.w3 = Wagon(self.can, 250, 30)
        self.w4 = Wagon(self.can, 370, 30)

    def coucou(self):
        "apparition de personnages dans certaines fenêtres"
        self.w1.perso(3)        # العربة الأولى, النافذة الثالثة
        self.w3.perso(1)        # العربة الثالثة, النافذة الأولى
        self.w3.perso(2)        # العربة الثالثة, النافذة الثانية
        self.w4.perso(1)        # العربة الرابعة, النافذة الأولى

class Wagon(object):
    def __init__(self, canev, x, y):
        "dessin d'un petit wagon en <x,y> dans le canevas <canev>"
        # تخزين برامترات في متغيرات المثيل :
        self.canev, self.x, self.y = canev, x, y
        # rectangle de base : 95x60 pixels :
        canev.create_rectangle(x, y, x+95, y+60)
        # المستطيل الأساسي : 95×60 بيكسل :
        for xf in range(x+5, x+90, 30):
            canev.create_rectangle(xf, y+5, xf+25, y+40)
        # عجلتين نصف قطرها يساوي 12 بكسل :
        cercle(canev, x+18, y+73, 12)
        cercle(canev, x+77, y+73, 12)

    def perso(self, fen):
        "apparition d'un petit personnage à la fenêtre <fen>"
        # حساب إحداثيات مركز كل نافذة :
        xf = self.x + fen*30 -12
        yf = self.y + 25
        cercle(self.canev, xf, yf, 10)      # الوجه
        cercle(self.canev, xf-5, yf-3, 2)   # العين اليسرى
        cercle(self.canev, xf+5, yf-3, 2)   # العين اليمنى
        cercle(self.canev, xf, yf+5, 3)     # الفم

app = Application()
app.mainloop()
```

تعليقات

- الأسطر من 3 إلى 5 : نحن نخطط لرسم سلسلة من الدوائر الصغيرة. وهذه الدالة الصغيرة تسهل عملنا من خلال تعريف الدوائر من خلال وسطها ونصف قطرها .
- الأسطر من 7 إلى 13 : تم صنع الصنف الرئيسي لبرنامجنا بالاشتقاق عن صنف النوافذ **Tk()** التي تم إستدعائها من وحدة **tkinter**.[70] كما شرحنا في الفصل السابق، منشئ الصنف المشتق يقوم بتنشيط بنفسه منشئ الصنف الأصل، ويمرر له مرجع المثيل كأول برامتر. الأسطر من 10 إلى 13 تستخدم لوضع اللوحة والأزرار في أماكنها .
- الأسطر من 15 إلى 20 : هذه الأسطر تقوم بتمثيل 4 كائنات-عربات، المنتجة من الصنف المقابل. ويمكننا البرمجة بأكثر أناقة بمساعد حلقة وقائمة، لكننا لا نريد أن نتعب أنفسنا بالتفسيرات الغير الضرورية. نحن نريد وضع كائنات-عرباتنا في اللوحة، في مواقع محددة : لذا يجب علينا أن نمرر بعض المعلومات لمنشئ لهذه الكائنات : على الأقل مرجع اللوحة، والإحداثيات المطلوبة. وهذه الاعتبارات تكون تلمبح عندما نعرف صنف **Wagon()**، نبتعد قليلا، يجب علينا أن نربط مع أسلوب منشئها عدد مساوي من البرامترات من أجل الحصول على هذه البرامترات.
- الأسطر من 22 إلى 27 : يتم استدعاء هذا الأسلوب عندما ننشط الزر الثاني. وهي تستدعي بنفسها الأسلوب **perso()** لبعض أجسام- العربات، مع برامترات مختلفة، لإظهار الأشخاص في نوافذ معينة. هذه الأسطر القليلة من الكود تظهر لك كيفية كائن يمكن التواصل مع كائنات أخرى، وذلك عن طريق أساليبه. وبعد هذه الالية المركزية للبرمجة بواسطة الكائنات :

> الكائنات هي كيانات برمجية التي تتفاعل من خلال تبادل الرسائل وأساليبها .

من الناحية المثالية، يجب على الأسلوب **coucou()** أن يشمل بعض التعليمات الإضافية، التي من شأنها أن تحقق أولا ما إذا كانت الكائنات-العربات موجودة بالفعل أو لا، قبل أن تسمح بتفعيل أساليبها. نحن لن نضع هذا النوع من الحماية للحفاظ على أكبر قدر ممكن من البساطة، لكن هذه نتيجة التسلسل التي لا تستطيع بها تفعيل الزر الثاني قبل الأول .

- الأسطر 29 و 30 : لا يشتق الصنف **Wagon()** من أي صنف آخر، لأنه صنف من الكائنات الرسومية، ومع ذلك يجب علينا أن نجلب البرامترات من المنشئ من أجل الحصول على مرجع اللوحة سنضع بها الرسوم، بالإضافة إلى إحداثيات هذه الرسوم. في التجاربك يمكنك من هذا التمرين، إضافة المزيد من البرامترات مثل : حجم الرسم، التوجيه، اللون، السرعة ...إلخ.

70سوف نرى لاحقا أن *tkinter* يسمح أيضا ببناء نافذة رئيسية لتطبيق باشتقاق من صنف لويدجت (في معظم الأحيان ستكون بالاشتقاق من ويدجت `Frame()`). النافذة التي تشمل هذا الويدجت سيتم إضافتها تلقائيا (انظر إلى صفحة 225).

- الأسطر من 31 إلى 51 : هذه الأسطر تتطلب القليل من التعاليق. الأسلوب **perso()** لديه برامتر الذي يشير إلى 3 النوافذ التي يجب أن يظهر بها الأشخاص الصغار. هنا أيضا ليس لدينا حماية : يمكنك استدعاء هذا الأسلوب مع برامتر يساوي 4 أو 5، على سبيل المثال، وهذا يحدث أشياء خاطئة .
- الأسطر 53 و 54 : لهذا التطبيق، على عكس سابقا، فضلنا الفصل بصنع الكائن **app**، ويبدأ من خلال استدعاء **mainloop()**ـ في هذين التعلمتين المنفصلتين (كمثال). يمكنك أيضا تقليل هتين التعليمتين في تعليمة واحدة، والتي ستكون : `Application().mainloop()`، وبالتالي وفرنا متغير .

تمارين

6.13 اجعل السكريبت الذي وصفناه في الأعلى مثاليا، بإضافة برامتر اللون إلى منشئ الصنف **Wagon()**ـ التي سوف تحدد لون مقصورة العربة. اجعل النوافذ سوداء في البداية، والعجلات رمادية (لتحقيق هذا الهدف، أضف أيضا برامتر اللون إلى دالة **cercle()**).

في صنف **Wagon()**ـ أضف أيضا الأسلوب **allumer()**، والذي سيستخدم لتغيير لون لثلاثة نوافذ (مبدئيا سوداء) إلى الأصفر، لمحاكاة الإضاءة الداخلية.

أضف زر إلى النافذة الرئيسية، التي تشغل الأضواء. حسن الدالة **cercle()** لتلوين وجوه الأشخاص الصغار إلى اللون الوردي (pink)، أعينهم وأفواههم إلى اللون الأسود، ومثل الكائنات-العربات مع ألوان مختلفة .

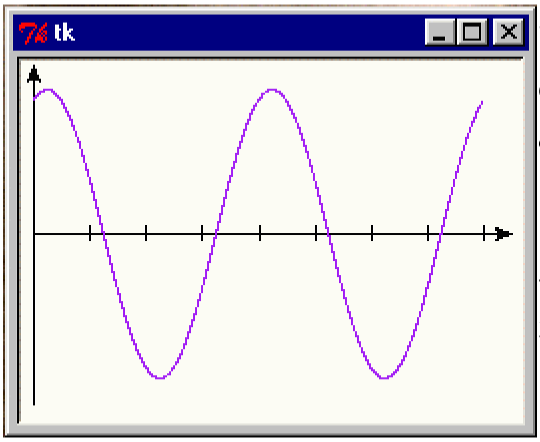
7.13 أضف إلى البرنامج السابق، بحيث يمكننا استخدام أي زر في الإضطراب، على الرغم من أن هذا سيؤدي إلى خطأ أو تأثير غريب .

مخطط الذبذبات : ودجة مخصصة

المشروع الذي سنقوم به الآن سيقودنا خطوة إلى الأمام. سنبني صنف ودجة جديد، والذي سيكون من الممكن إضافته إلى مشاريعنا القادمة مثل أي ودجة عادية. مثل الصنف الأساسي في التمرين السابق، هذا الصن الجديد سيتم بناءه بالاشتقاق لصنف tkinter الموجود مسبقا.

هذا الموضوع لهذا التطبيق سوف نستلهم بعض منه من درس الفيزياء .

Pour rappel, un mouvement vibratoire harmonique se définit comme étant la projection d'un mouvement circulaire uniforme sur une droite. Les positions successives d'un mobile qui effectue ce type de mouvement sont traditionnellement repérées par rapport à une position centrale : on les appelle alors des *élongations*. L'équation qui décrit l'évolution de l'élongation d'un tel mobile au cours du temps est toujours de la forme $e = A \sin(2\pi f t + \phi)$, dans laquelle ***e*** représente l'élongation du mobile à tout instant ***t***. Les constantes ***A, f*** et *φ* .désignent respectivement l'*amplitude*, la *fréquence* et la *phase* du mouvement vibratoire



الهدف من هذا المشروع توفير أداة بسيطة لعرض هذه المفاهيم المختلفة، وهي نظام لعرض تلقائيا رسومات إطالة/الوقت. يستطيع المستخدم أن يختار بحرية قيم البرامترات A و f و φ ويجب عليه أن يراعي المنحنيات الناتجة عن ذلك.

يرجى ترميز السكريبت بالأسفل وحفظه في ملف باسم **oscillo.py**. أنت تعرف أن الوحدة الحقيقية تحتوي على صنف (يمكنك إضافة أصناف أخرى في وقت لاحق في نفس الوحدة) .

```
from tkinter import *
from math import sin, pi

class OscilloGraphe(Canvas):
    "Canevas spécialisé, pour dessiner des courbes élongation/temps"
    def __init__(self, boss =None, larg=200, haut=150):
        "Constructeur du graphique : axes et échelle horiz."
        # منشئ ودجة الأصل :
        Canvas.__init__(self)                          # استدعاء منشئ
        self.configure(width=larg, height=haut)        # الصنف الأصل
        self.larg, self.haut = larg, haut                       # حفظ
        # مسار محاور المرجع :
        self.create_line(10, haut/2, larg, haut/2, arrow=LAST)  # axe X
        self.create_line(10, haut-5, 10, 5, arrow=LAST)         # axe Y
        # رسم سلم مع 8 درجات :
        pas = (larg-25)/8.          # فترات سلم أفقي
        for t in range(1, 9):
            stx = 10 + t*pas        # +10 للبدأ من المنشئ
            self.create_line(stx, haut/2-4, stx, haut/2+4)

    def traceCourbe(self, freq=1, phase=0, ampl=10, coul='red'):
        "tracé d'un graphique élongation/temps sur 1 seconde"
        curve =[]                       # قائمة الإحداثيات
        pas = (self.larg-25)/1000.      # الموقع لـ 1 ثانية × السلم
        for t in range(0,1001,5):       # التي تنقسم إلى 100 ميلي ثانية
            e = ampl*sin(2*pi*freq*t/1000 - phase)
            x = 10 + t*pas
            y = self.haut/2 - e*self.haut/25
            curve.append((x,y))
        n = self.create_line(curve, fill=coul, smooth=1)
        return n                        # n = الرقم التسلسلي للمدول

#### كود لتجربة الصنف : ####

if __name__ == '__main__':
    root = Tk()
    gra = OscilloGraphe(root, 250, 180)
    gra.pack()
    gra.configure(bg ='ivory', bd =2, relief=SUNKEN)
    gra.traceCourbe(2, 1.2, 10, 'purple')
    root.mainloop()
```

المستوى الرئيسي للسكريبت يتكون من السطر 35 إلى السطر 41.

كما سبق أن أوضحنا في الصفحة202، أسطر الكود التي بعد التعليمة `if __name__ == '__main__':` لن يتم تنفذها إذا كان السكريبت تم إستدعائه كوحدة في تطبيق آخر. فإذا شغلت هذا السكريبت كبرنامج رئيسي سيتم تنفيذ هذه التعليمات. وبالتالي لدينا الية مثيرة للإهتمام، فهي تسمح لنا بدمج التعليمات الاختبار في إطار وحدة، حتى لو كانت معدة ليتم إستيرادها إلى سكريبتات أخرى.

قم إذا بتشغيل السكريبت بالطريقة العادية. يجب عليك أن تحصل على عرض يشبه الذي قمنا به في الصفحات السابقة .

تجربة

سوف نقوم بالتعليق على الأسطر المهمة للسكريبت. لكن لنبدأ أولا من خلال تجربة قليلا الصنف الذي صنعناه للتو.

إفتح إذا طرفيتك، وأدخل التعليمات بالأسفل مباشرة إلى سطر الأوامر :

```
>>> from oscillo import *
>>> g1 = OscilloGraphe()
>>> g1.pack()
```

بعد استدعاء صنف الوحدة **oscillo**، قمنا بتمثيل كائن أول **g1**، للصنف **OscilloGraphe()**.

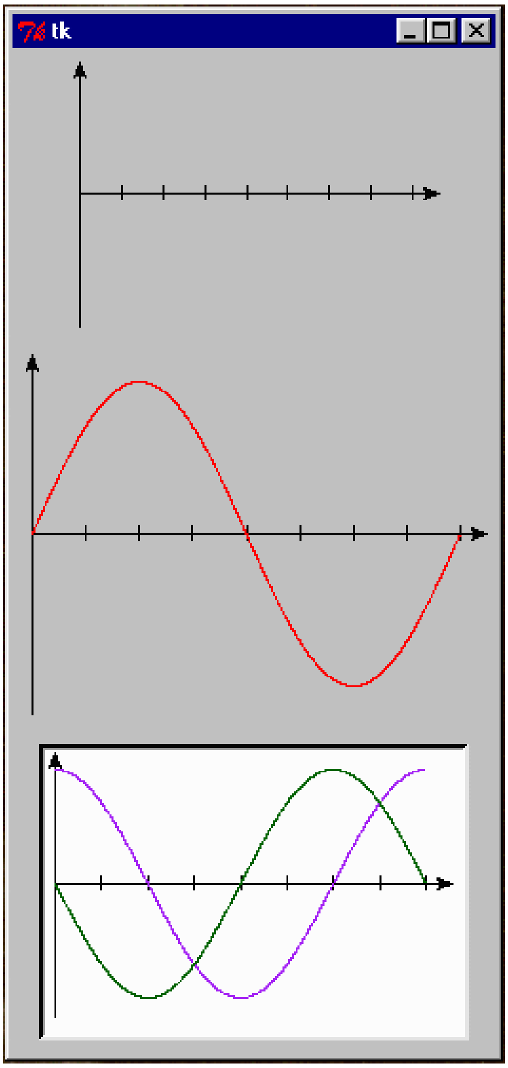
بما أننا لم نضع أي برامتر، الكائن لديه حجمه الافتراضي، تم تعريفه في منشئ الصنف. لاحظ أننا لم نقم بتعريف أولا نافذة الأصل لكي نضع بها الويدجات. tkinter يسامح هذا النسيان وهو وفره لنا تلقائيا !

```
>>> g2 = OscilloGraphe(haut=200, larg=250)
>>> g2.pack()
>>> g2.traceCourbe()
```

لهذه التعليمات، صنعنا ودجة ثاني من نفس الصنف، وهذه المرة مع تحديد أبعادها (الطول والعرض، لا يهم الترتيب).

ثم قمنا بتفعيل الأسلوب **traceCourbe()** المرتبط بهذا الودجة. لأننا لم نوفر لها أي برامتر، ويظهر الشرط الموجب الذي يتوافق مع القيم الافتراضية للبرامترات A و f و φ .

```
>>> g3 = OscilloGraphe(larg=220)
>>> g3.configure(bg='white', bd=3, relief=SUNKEN)
>>> g3.pack(padx=5,pady=5)
>>> g3.traceCourbe(phase=1.57, coul='purple')
>>> g3.traceCourbe(phase=3.14, coul='dark green')
```



لفهم تكوين الودجة الثالثة، يجب أن تتذكر الصنف **OscilloGraphe()** تم صنعه باشتقاق من الصنف **Canvas()**. وهو يرث جميع خصائصه، والذي يتيح لنا اختيار لون الخلفية، والحدود ... إلخ، باستخدام نفس البرامترات التي وضعناها عندما كونا اللوحة.

ثم قمنا بعرض قطعتين متعاقبتين، وذلك عن طريق استدعاء الأسلوب **traceCourbe()**، مرتين، والتي نقدم البرامترات للتطور واللون.

تمرين

8.13 قم بصنع ودجة رابعة، بقياس 400 × 300، بلون خلفية أصفر، وقم بإظهار العديد من المنحنيات الموافقة لترددات مختلفة.

لقد حان الوقت لتحليل هيكل الصنف الذي سمحنا له بتمثيل هذه الويدجات. سوف نقوم الآن بحفظ هذا الصنف في وحدة تدعى **oscillo.py** (انظر للصفحة 214).

المواصفات

نحن نريد الآن تعريف صنف لودجة جديدة، قادر على إظهار تلقائيا رسوم بيانية إطالة/وقت متوافقة مع حركات الذبذبات.

يجب على هذه الودجة أن تكون قادرة على صنع مثيل في أي وقت. ويجب عليها أن تظهر محورين ديكارتي X و Y مع أسهم. ويمثل المحور X مرور الوقت بالثانية الواحدة وسوف تكون مجهزة بـ 8 فترات.

و سوف يتم ربط الأسلوب **traceCourbe()** بهذه الودجة. وهذا قد يكون سبب الرسوم البيانية إستطالة/الزمن لحركة الإهتزازية، والتي يجب علينا تقديم النواتر (هزة تتروح ما بين 0.25 إلى 10 هرتز) ومرحلة (ما بين 0 و **2π** راديان) والسعة (بين 1 إلى 10 : نطاق تعسفي).

التطبيق

- السطر 4 : تم صنع الصنف **OscilloGraphe()** بالانشقاق من الصنف **Canvas()**. وهو يرث جميع خصائصه : يمكن تكوين كائنات لهذا الصنف الجديد باستخدام العديد من الخيارات المتاحة بالفعل للصنف **Canvas()**.
- السطر 6 : إن الأسلوب المنشئ يستخدم ثلاثة برامترات، وكلها اختيارية، لأن لكل واحدة منها قيمة افتراضية. يستخدم البرامتر **boss** لتلقي الإشارة فقط لمرجع نافذة (انظر الأمثلة أدناه). البرامترا **larg** و **haut** (للعرض والإرتفاع) لتعيين قيم لخيارات **width** و **height** للوحة الأصل، في لحظة تمثيله
- لقد قمنا بتفعيل المنشئ للصنف **Canvas()** في السطر 9، وقمنا بإضافة خيارين له في السطر 10. لاحظ أننا كنا نستطيع أن نختصر هذان السطران في سطر واحد وهو :

```
Canvas.__init__(self, width=larg, height=haut).
```

و كما شرحنا (انظر إلى الصفحة 198)، يجب علينا تمرير للمنشئ مرجع المثيل (**self**) كبرامتر أول.

- السطر 11 : إنه من الضروري حفظ البرامترات **larg** و **haut** في متغير مثيل، لأننا يجل علينا الوصول إليه أيضا في الأسلوب **traceCourbe()**.
- السطران 13 و 14 : لرسم محاور X و Y، سوف نستخدم البرامترات **larg** و **haut**، ويتم وضع هذه المحاول تلقائيا على الأبعاد. يستخدم الخيار **arrow=LAST** لعرض سهم صغير في نهاية كل خط .
- الأسطر من 16 إلى 19 : لرسم مقياس أفقي، نبدأ من خفض 25 بكسل من عرض المتاح، لكي يتم تشكيل مساحات في كل الجانبين. ثم نقسمها إلى 8 أقسام، وهو تصور شكل عمودي لثمانية خطوط صغيرة .

• السطر 21 : يمكن استدعاء الأسلوب **traceCourbe()** مع أربعة برامترات. كل واحد منها يمكن حذفه، لأن كل البرامترات لديها قيم افتراضية. ويمكن توفير البرامترات في أي ترتيب ، كما شرحنا في الصفحة 79.

• الأسطر من 23 إلى 31 : لرسم منحنى المتغير **t** يتم القيم من 0 إلى 1000، ويحسب كل مرة إستطالة **e** المقابلة، بمساعدة الصيغة النظرية (السطر 26). تم العثور على أزواج القيم **t** و **e** وتم تحجيمهم وتحويلهم إلى إحداثيات **x, y** الأسطر 27 و 28، ثم تراكمت في قائمة **curve**.

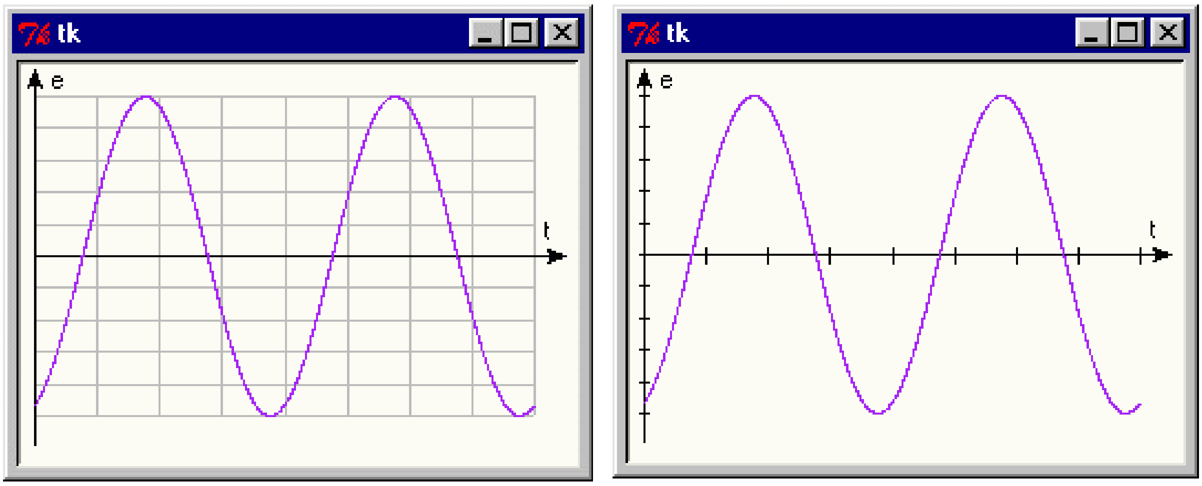
• الأسطر 30 و 31 : يرسم الأسلوب **create_line()** المنحنى المقابل في عملية واحدة، ويقوم بإرجاع رقم ترتيب للكائن الجديد الذي تم تمثيله في اللوحة (و هذا رقم الترتيب سوف يسمح لنا بالوصول إليه بعد ذلك : لمسحه، على سبيل المثال). الخيار `smooth =1` يحسن مظهر النهائي بتجانسه.

تمارين

9.13 عدل السكريبت بحيث يصبح المحور هو المرجع العمودي يضم إليه أيضا مقياس، مع 5 أجزاء de et d'autre l'origine.

10.13 مثل ودجات الصنف **Canvas()** الذي يشتق منه، ودجة الهاص بك يمكنه دمج مؤشرات نصية. ببساطة استخدم الأسلوب **create_text()**. هذا الأسلوب ينتظر على الأقل ثلاثة برامترات. الإحداثيات **x , y** لمكان الذي تريد أن تظهر نصك فيه ثم النص الذي تريد إظهاره بطبيعة الحال. ويمكن تمرير برامترات أخرى بشكل خيارات، لتحديد على سبيل المثال الخط والحجم. لنرى كيف يعمل هذا، أضف مؤقتا السطر التالي في منشئ الصنف **OscilloGraphe()**، ثم قم بإعادة تشغيل السكريبت :

```
self.create_text(130, 30, text = "Essai", anchor =CENTER)
```



استخدم هذا الأسلوب لإضافة إلى ودجة المؤشرات التالية القصوى لمحاور المرجع: **e** (للإستطالة) علـى طـور محـور العمودي، و **t** (للوقت أو الزمن) على طول المحور الأفقي. قد تبدو النتيجة على الشكل الأيسر .

11.13 ومرة أخرى، يمكنك إكمال الودجة الخاصة بك بإظهار شبكة المرجعـ التي هـي شرائط بسيطة على طول المحاور، لتأكد من أن هذه الشرطات لن تكون مرئية أكثر من اللازم، يمكنك تلوين ملامحها باللون الرمادي (الخيار fill = ''grey'')، كما في الشكل بالأعلى.

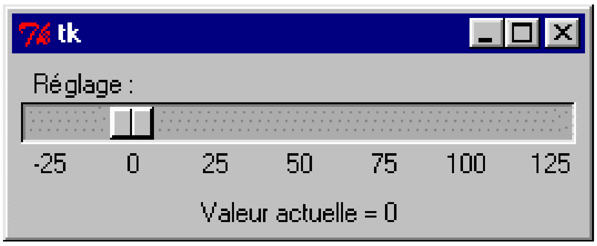
12.13 أكمل الويدجت الخاص بك بإظهار الأرقام.

المؤشرات . ودجة مركبة

في التمرين السابق، قمت بصنع نوع جديد من الويدجات التي قمت بحفظها في وحدة **oscillo.py**. حافظ على هذه الوحدة سوف تنضم قريبا لمشروع أكبر تعقيدا.

في الوقت الراهن، سوف تقوم بصنع ودجة أخرى، وهذه المرة أكثر تفاعلا. ستكون نوع من أنواع لوحات التحكم تحتوي على ثلاثة متزلجات وخانة اختيار. مثل سابقتها، يهدف هذا الودجة لإعادة استخدامه في تطبيق تجميعي .

عرض ودجة المقياس (Scale)



دعونا نبدأ أولا من خلال إستكشاف ودجة القاعدة، والتي لم نستخدمها حتى الآن : الودجة **Scale** يشبه المتزحلق الذي يتحلق أمام مقياس. يسمح للمستخدم اختيار سرعة القيمة بأي برامتر.

السكريبت الضغير بالأسفل يوضح لك كيف وضع برامتر واستخدامها في نافذة :

```
from tkinter import *

def updateLabel(x):
    lab.configure(text='Valeur actuelle = ' + str(x))

root = Tk()
Scale(root, length=250, orient=HORIZONTAL, label ='Réglage :',
      troughcolor ='dark grey', sliderlength =20,
      showvalue =0, from_=-25, to=125, tickinterval =25,
      command=updateLabel).pack()
```

```
lab = Label(root)
lab.pack()

root.mainloop()
```

هذه الأسطر لا تتطلب الكثير من التعليق.

يمكنك إنشاء ودجات **Scale** بأي حجم كان (الخيار **length**)، في إتجاه أفقي (كما في مثالنا) أو عمودي (الخيار **orient = VERTICAL**).

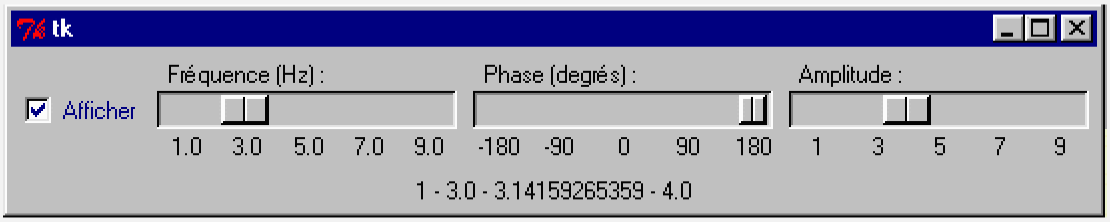
الخيار **from_** (انتبه : لا تنسى الرمز "تحت السطر"، لأن هذا إلزامي لكي لا يتم الخلط بين هذه الكلمة وكلمة **from** المحجوزة!) و **to** لضبط النطاق. يتم تعريف الفترة بين الأرقام في الخيار **tickinterval**، إلخ.

و الدالة تحدد في خيار **command** بإنه يستدعي تلقائيا في كل مرة عندما ستم وضع المؤشر والمكان الحالي للمؤشر حسب المقيار يتم تمريره كبرامتر. لذا فهو من السهل جدا استخدام هذه القيمة لتعيين أي معالجة. انظر على سبيل المثال إلى البرامتر **x** لدالة **updateLabel()**، في مثالنا.

الودجة **Scale** هي واجهة بديهية للغاية وتوفر للمستخدمي برامجك العديد من الإعدادات. سوف نقوم الآن بدمج عدة نسخ منها في صنف ودجة جديد : لوحة تحكم لاختيار التردد، والحركة والطور لحركات الذبذبات، ثم سنقوم بعرض ريم بياني إستطالة/الوقت بمساعدة ودجة **oscilloGraphe** الذي درسناه في الصفحات السابقة .

بناء لوحة تحكم بثلاثة متزلجات/مؤشرات

مثل سابقتها، السكريبت الموصوف بالأسفل يجب أن يتم حفظه في وحده، والذي يجب أن يتم تسميته بـ **curseurs.py**. الأصناف التي قمت بحفظها سيتم إعادة استخدامها (بالاستدعاء) في تطبيق مركب والذي سوف نتحدث عنه في وقت لاحق. ولاحظوا أننا نستطيع تقصير الكوج بالأسفل لطرق مختلفة (سوف نتحدث عنها لاحقا[71]. نحن لم نحسن الكود، لأن هذا يتطلب دمج مفهومات إضافية (العبارة lambda)، والتي نفصل أن نتجنبها في الوقت الحالي. أنت تعرف بالفعل أن أسطر الكود الموجود في أسفل السكريبت لاختبار عملها. سوف تحصل على نافذة مثل هذه :




```
from tkinter import *
from math import pi
```

71 يمكنك بالطبع حفظ العديد من الأصناف في نفس الوحدة .

```

class ChoixVibra(Frame):
    """Curseurs pour choisir fréquence, phase & amplitude d'une vibration"""
    def __init__(self, boss =None, coul ='red'):
        Frame.__init__(self)        # منشئ الصنف الأصل
        # تهيئة بعض سمات المثيل :
        self.freq, self.phase, self.ampl, self.coul = 0, 0, 0, coul
        # متغيرة حالة خانة الاختيار :
        self.chk = IntVar()                 # 'كائن-متغير' tkinter
        Checkbutton(self, text='Afficher', variable=self.chk,
                    fg = self.coul, command = self.setCurve).pack(side=LEFT)
        # تعريف ثلاثة ويدجات من نوع متزلجات :
        Scale(self, length=150, orient=HORIZONTAL, sliderlength =25,
              label ='Fréquence (Hz) :', from_=1., to=9., tickinterval =2,
              resolution =0.25,
              showvalue =0, command = self.setFrequency).pack(side=LEFT)
        Scale(self, length=150, orient=HORIZONTAL, sliderlength =15,
              label ='Phase (degrés) :', from_=-180, to=180, tickinterval =90,
              showvalue =0, command = self.setPhase).pack(side=LEFT)
        Scale(self, length=150, orient=HORIZONTAL, sliderlength =25,
              label ='Amplitude :', from_=1, to=9, tickinterval =2,
              showvalue =0, command = self.setAmplitude).pack(side=LEFT)

    def setCurve(self):
        self.event_generate('<Control-Z>')

    def setFrequency(self, f):
        self.freq = float(f)
        self.event_generate('<Control-Z>')

    def setPhase(self, p):
        pp =float(p)
        self.phase = pp*2*pi/360        # تحويل الدرجة -> راديان
        self.event_generate('<Control-Z>')

    def setAmplitude(self, a):
        self.ampl = float(a)
        self.event_generate('<Control-Z>')

#### كود لتجربة الصنف : ###

if __name__ == '__main__':
    def afficherTout(event=None):
        lab.configure(text = '{0} - {1} - {2} - {3}'.\
                      format(fra.chk.get(), fra.freq, fra.phase, fra.ampl))
    root = Tk()
    fra = ChoixVibra(root,'navy')
    fra.pack(side =TOP)
    lab = Label(root, text ='test')
    lab.pack()
    root.bind('<Control-Z>', afficherTout)
    root.mainloop()
```

هذه لوحة التحكم تسمح للمستحدمين بضبط القيم البرامترات المعينة (تردد وطور وسعة)، ومن ثم يمكن استخدامه لعرض رسم بياني إطالة/الزمن في ودجة للصنف **OscilloGraphe()** الذي قمنا بصنعه سابقا، كما وضحنا في التطبيق .

تعليقات

- السطر 6 : يستخدم الأسلوب المنشئ برامتر الاختياري **coul**. هذا البرامتر يسمح باختيار لون غرافيك للوحة التحكم للودجة. البرامتر **boss** لتلقي مرجع النافذة الأصل الممكنة (سوف نراها لاحقا).
- السطر 7 : تفعيل منشئ الصنف الأصل (لوراثة وظائفه).
- السطر 9 : بيان ببعض المتغيرات المثيل. وسيتم تحديد قيمهم بواسطة الأساليب في الأسطر 29 إلى 49 (معالجة الأحداث).
- السطر 11 : هذه التعليمة تقوم بتمثيل كائن من صنف **IntVar()**- والذي هو جزء من وحدة tkinter وهو مشابه للأصناف **StringVar**، **DoubleVar()** و **BooleanVar()**. كل هذه الأصناف تسمح بتعريف متغيرات tkinter، والتي هي في الواقع كائنات، لكنها تتصرف مثل المتغيرات في ويدجات tkinter (انظر لاحقا). وبالتالي المرجع في **self.chk** يحتوي على ما يعادل متغير من نوع صحيح، في شكل مستخدم من قبل tkinter. للوصول إلى قمته من بيثون، يجب استخدام الأساليب الخاصة بهذا الكائن : الأسلوب **set()** يسمح بتعيين قيمة، والأسلوب **get()** يسمح بإسترجاعها (سوف نراه في السطر 47).
- السطر 12 : الخيار **variable** لمتغير **checkbutton** يقوم بربط المتغير tkinter الذي قمنا بتعريفه في السطر السابق. نحن لا يمكننا أن نقوم برمجع مباشر لمتغير عادي في تعريق ودجة tkinter، لأن tkinter كتب بلغة لا تستخدم نفس إتفاقيات بيثون لتنسيق المتغيرات. الكائنات تم بنائها من أصناف متغيرات tkinter ضرورية لضمان الواجهة.
- السطر 13 : الخيار **command** يشير إلى أسلوب الذي يجب على النظام استدعاءه عندما يقون المستخدم بالنقر من الفأرة على خانة الاختيار.
- الأسطر من 14 إلى 24 : هذه الأسطر تقوم بتعريف ثلاثة ويدجات متزلجات، في ثلاثة تعليمات متشابهة. وسيكون من الأفضل إختصار هذه تعليمات إلى واحدة فقط، يتم تكرارها ثلاثة مرات بمساعدة حلقة. وهذا يتطلب مفهوم لم أقم بشرحه بعد (الدالات أو العبارات lambda)، وتعريف معالج الأحداث الذي يتم ربطه بهذه الويدجات ستصبح أكثر تعقيدا. حتى هذا الوقت سوف تبقى التعليمات منفصلة : سنحاول تحسينها لاحقا.
- الأسطر من 26 إلى 40 : الويدجات الأربعة تم تعريفهم في الأسطر سابقة لدى كل واحدة منها خيار **command**. لكل واحدة منها، أسلوب استدعاء الخيار **command** مختلف : مربع الاختيار يفعل الأسلوب **setCurve()**- فيقوم المتزلج الأول بتنشيط الأسلوب **setFrequency()**، والمتزلج الثاني يفعل الأسلوب **setPhase()**، وأما الثالث فيفعل الأسلوب **setAmplitude()**. لاحظ أن الخيار **command** لودجات **Scale** تمرر برامتر للأسلوب

المرتبط (موضع الحالي للمتزلج)، لذا فقط الخيار **command** لا يقوم بتمرير شيء في حالة الودجة **Checkbutton**.

هذه الأساليب الأربعة (و التي هي معالجة الأحداث التي تم إنشاؤها بواسطة خانة و 3 متزلجات) وتسبب كل واحدة منها مهمة عنصر جديد[72]، وذلك باستدعاء الأسلوب **event_generate()**.

عندما يتم استدعاء هذا الأسلوب، يقوم بيثون بإرجاع لنظام التشغيل نفس رسالة-العناصر التي سوف تظهر إذا ضغط المستخدم على <Ctrl> و<Maj> و<Z> في وقت واحد من لوحة المفاتيح.

و سوف نقوم بإظهار رسالة-العناصر خاصة جدا، الذي تقوم بكشف ومعالجة بواسطة معالج الأحداث المترابط بودجة آخر (انظر للصفحة التالية). بهذه الطريقة، وضعنا في المكان نظام الصحيح للإتصال بين الويدجات : في كل مرة يقوم بها المستخدم بتنفيذ إجراء على لوحة التحكم الخاصة بنا، فإنه ينشئ حدث معين مما يدل على العمل لإنتباه الحاجيات الأخرى .

> يمكننا اختيار تركيبة مفاتيح أخرى (أو نوع آخر من الأحداث). ولكم فإن هنالك فرصة ضئيلة جدا أن يقوم المستخدم باستخدامها عندما يستخدم برنامجنا. ومع ذلك، يمكننا أن ننتج مثل هذا الحدث بأنفسنا كما، كتجربة، عندما يحي الوقت للتحقق من معالج هذا الحدث، والذي سوف نعيد استخدامه في وقت لاحق .

• الأسطر من 42 إلى 54 مثلما فعلنا لـ **oscillo.py**، أكملنا هذه الوحدة الجديدة ببضعة أسطر في المستوى الرئيسي. هذه الأسطر تسمح باختبار التشغيل الصنف : وهي لا تعمل إلا لو شغلت الوحدة مباشرة، كتطبيق مستقل. تأكد من استخدام هذه التقنية في وحدات الخاصة بك، لأن هذه الممارسة جيد في البرمجة : مستخدم وحدات البناء يمكن في الواقع قد (يعيد) إكتشاف دالاتها بسهولة جدا (عن طريق تشغيل) وكيفية استخدامها (من خلال تحليل أسطر تعليماتها البرمجية).

في هذه أسطر للاختبار، قمنا ببناء نافذة أساسية **root** والتي تحتوي على ودجتان : ودجة لصنف جديد **ChoixVibra()** وودجة للصنف **Label()**.

في السطر 52، قمنا بربط النافذة الرئيسية بمعالج الأحداث : جميع الأحداث من نوع محدد سوف تؤدي إلى استدعاء الدالة **afficherTout()**.

---

72 في الواقع، يجب علينا أن نسميه رسالة (و الذي هو في حد ذاته إعلام حدث) . يرجى قراءة الشرح في الصفحة 88 : برامج يتم التحكم بها بواسطة الأحداث.

هذه الدالة هي معالج الأحداث الخاص، والتي يتم تطبقها في كل مرة عندما يكون نوع الحدث <Maj-Ctrl-Z> تم كشفه من قبل نظام التشغيل.

كما شرحنا أعلاه، علينا أن نضمن أن يتم صنع مثل هذه الأحداث من قبل كائنات الصنف **ChoixVibra()**ـ في كل مرة قام المستخدم بتعديل حالة أو أخرى من المتزلجات الثلاثة، أو من خانة الاختيار.

- صمم فقط لاختبار الدالة **afficherTout()** لا تفعل شيئا لكنها تتسبب في عرض قيم المتغيرات المرتبطة بويدجاتنا الأربعة، بإعادة تكون خيار **text** لودجة من صنف **Label()**.
- السطر 47، التعبير `fra.chk.get()` : لقد رأينا أعلاه أن متغيرـ تخزين حالة كائن خانة الاختيار هو كائن-متغيرـ tkinter. بيثون لا يمكنه قراءة مباشرة محتوى لمتغير مثل هذا، والذي هو في الواقع واجهة-كائن. لاستخراج القيمة، يجب علينا استخدام الأسلوب الخاص لهذا الصنف من الكائنات : الأسلوب **get()**.

نشر الأحداث

آلية الإتصال التي تم وصفها في الأعلى تتوافق مع تدرج أصناف الويدجات. ستلاحظ أنه يرتبط بأسلوب الذي يطرح هذا الحدث المرتبطه مع ودجة التي نحن نريد تعرف الصنف، عن طريق self. عامة، رسالة-الأحداث هي في الواقع مرتبطو بودجة خاصة (على سبيل المثال، يتم ربط ضغطة الفأرة على زر مع هذا الزر)، وهذا يعنيـ أن النظام التشغيل سوف يفحص أولا ما إذا كان هناك معالج لهذا النوع من حدث، الذي يرتبط أيضا مع هذا الودجة. فإذا وجد، فسوف ينشطه، وينشرـ رسالة توقف. وإلا، رسالة-الحدث هي التي تعرض على لتوالي على ودجة الأصل، برتيب تدرجيا، إلى أن يجدها معالج الأحداث، أو حتى يتم الوصول إلى النافذة الرئيسية.

الأحداث الموافقة لضغطات على لوحة المفاتيح (مثل ضغطة `<Maj-Ctrl-Z>` التي تم استخدامها في تمريننا) يتم دائما شحنها مباشرة إلى النافذة الرئيسية للتطبيق. في مثالنا، معالج الأحدصل لهذا العنصر يجب عليه يرتبط مع نافذة **root**.

تمارين

13.13 الودجة الجديد الخاص بك يرث جميع خصائص الصنف **Frame()**. يمكنك إذا تعديل مظهره عن طريق تعديل خيارات الافتراضية لهذا الصنف، بمساعدة الأسلوب **configure()**. حاول على سبيل المثال أن تقوم بإحاطة لوحة التحكم بأربعة بيكسلات بعد إظهار الأخدود (`bd = 4, relief = GROOVE`). فإذا لم تفهم ما يجب عليك فعله، إستوحي من السكريبت **oscillo.py** (السطر العاشر).

14.13 إذا قمنا بتعيينـ القيمة 1 لخيار **showvalue** في ويدجات **Scale()**ـ فسيتم عرض نطاق موقع الدقيق للمتزحلق. لذا قم بتفعيل هذه الميزة للمتزحلق الذي يتحكم في برامتر "phase - طور" .

15.13 الخيار **troughcolor** لودجات **Scale()** تسمح بتعريق لون الشريحة. استخدم هذا الخيار للتأكد من أن لـون المتزلجات الثلاثة يتم استخدامهم كبرامتر لتمثيل ودجة جديد.

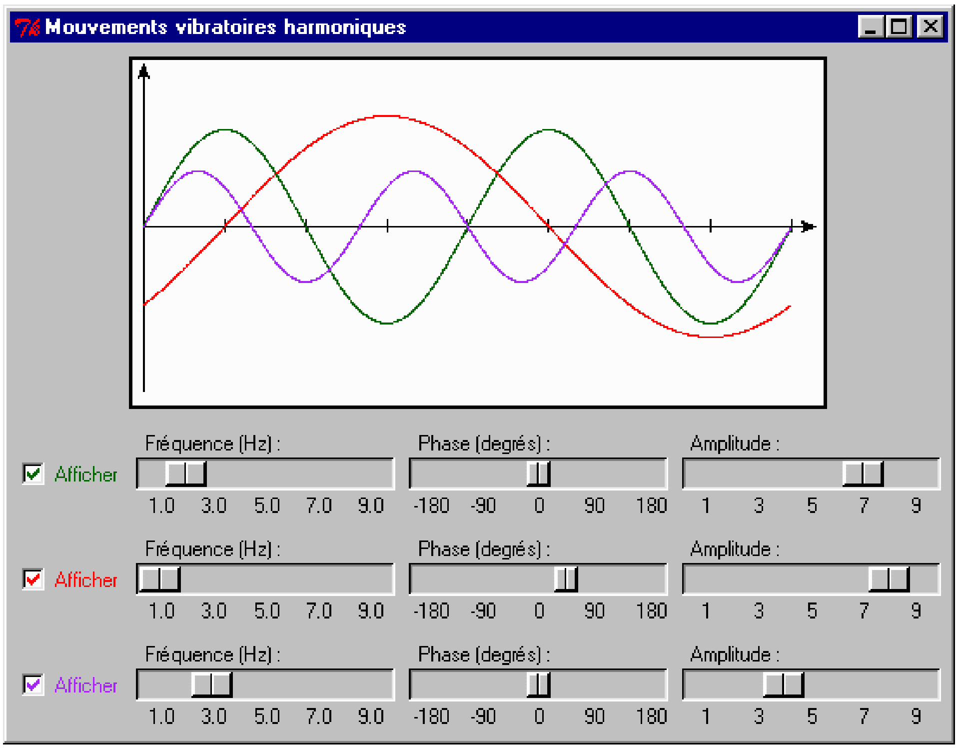
16.13 قم بتعديل السكريبت بحيث يتم إزالة ويدجات المتزلجات (الخياران **padx** و **pady** للأسلوب **pack()**).

دمج ويدجات المركبة في تطبيق تجميعي

في التمارين السابقة، قمنا ببناء صنفين ودجة جديدين : الودجة **OscilloGraphe()**، وهـي لوحة خاصة لرسم المخططات الذبذبات، والودجة **ChoixVibra()**، هو لوحة تحكم بثلاثة متزلجات يسمحون باختيار البرامترات الإهتزاز .

هو لوحة تحكم بثلاثة متزلجات يسمحون باختيار البرامترات الإهتزاز. هذه الويدجات متوفرة الآن في وحـدتي **oscillo.py** و **curseurs.py**[73].

سوف نقوم الآن باستخدامهم في تطبيق التركيبي : الودجة **OscilloGraphe()** سوف يقوم بعرض رسم، رسمين أو 3 رسوم لمخططات متذبذبة، بألوان مختلفة، كل واحدة منهم تحت تحكم الودجة **ChoixVibra()**.



السكريبت المقابل يصنع التالي .

73 و يمكننا أيضا جمع جميع الأصناف التي نصنعها في واحدة واحدة .

نلفت إنتباهكم إلى أن تشغيل التقنية لإحداث تحديث للعرض في لوحة من خلال حدث، في كل مرة يقوم المستخدم بتنفيذ أي حركة في مستوى في أي واحدة من لوحات التحكم.

تذكر أن التطبيقات مصممة لتعمل في واجهة المستخدم الرسومية وهذا يعني "البرنامج يتم التحكم به من خلال الأحداث" (انظر للصفحة 88).

الإعداد لهذا المثال، قررنا بشكل تعسفي بأن عارض الرسومات يسبب حدث معين، مماثل للذي يتم إنشاؤه بواسط نظام التشغيل عندما يقوم المستخدم بعمل أي حركة. في نطاق (كبير جدا) من الأحداث الممكنة، إخترنا واحدا الذي من غير المرجح أن يستخدم لأسباب أخرى، في حين برنامجنا يعمل : بتركيبة المفاتيح <Maj-Ctrl-Z>.

عندما قمنا بصنع صنف ودجة **ChoixVibra**()، وقمنا بدمج التعليمات اللازمة ليتم صنع حدث في كل مرة يقوم فيها المستخدم بتحريك أحدى المنزلجات أو تعديل حالة خانة الاختيار. سوف نقوم الآن بتعريف معالج لهذه الأحداث ونقوم بإدراجه في صنف جديد : الذي سنسميه **montreCourbes**() ويقوم بتحديث الشاشة. بالنظر لهذا الحدث هو ذات صلة به <enfoncement الضغطة>، ومع ذلك يجب أن نكتشف في مستوى النافذة الرئيسية للتطبيق .

```
1#  from oscillo import *
2#  from curseurs import *
3#
4#  class ShowVibra(Frame):
5#      """Démonstration de mouvements vibratoires harmoniques"""
6#      def __init__(self, boss =None):
7#          Frame.__init__(self)          # منشئ الصنف الأصل
8#          self.couleur = ['dark green', 'red', 'purple']
9#          self.trace = [0]*3            # قائمة المسارات (منحنيات لرسمها)
10#         self.controle = [0]*3         # قائمة لوحات التحكم
11#
12#         # تمثيل لوحة مع محاور X وY :
13#         self.gra = OscilloGraphe(self, larg =400, haut=200)
14#         self.gra.configure(bg ='white', bd=2, relief=SOLID)
15#         self.gra.pack(side =TOP, pady=5)
16#
17#         # تمثيل ثلاثة لوحات تحكم (متزيجلت)  :
18#         for i in range(3):
19#             self.controle[i] = ChoixVibra(self, self.couleur[i])
20#             self.controle[i].pack()
21#
22#         # تعيين الحدث الذي يقوم بعرض المسارات :
23#         self.master.bind('<Control-Z>', self.montreCourbes)
24#         self.master.title('Mouvements vibratoires harmoniques')
25#         self.pack()
26#
27#     def montreCourbes(self, event):
28#         """(Ré)Affichage des trois graphiques élongation/temps"""
29#         for i in range(3):
30#
31#             # أولا, إمسح المسار السابق (إن وجد) :
32#             self.gra.delete(self.trace[i])
33#
34#             # ثم, أرسم مسار جديد :
35#             if self.controle[i].chk.get():
36#                 self.trace[i] = self.gra.traceCourbe(
37#                                     coul = self.couleur[i],
38#                                     freq = self.controle[i].freq,
39#                                     phase = self.controle[i].phase,
40#                                     ampl = self.controle[i].ampl)
41#
42# #### كود لتجربة الصنف : ###
43#
44# if __name__ == '__main__':
45#     ShowVibra().mainloop()
```

تعليقات

- السطران 1 و 2 : نحن لسنا بحاجة إلى استدعاء وحدة tkinter : كم واحدة من هذه الوحدات تدعمه.
- السطر 4 : وبما أننا بدأنا بالتعرف على التقنيات الجديدة، قررنا إنشاء تطبيق نفسه كصنف ودجة جديد، مشتق من الصنف **Frame()** : حتى نتمكن من دمجة في وقت لاحق في مشاريع أخرى.
- الأسطر من 8 إلى 10 : لقد قمنا بتعريف بعض متغيرات المثيل (3 قوائم) : المنحنيات الثلاثة تصبح كائنات رسومية، والألون تم تعريفها مسبقا في قائمة **self.couleur** ; ويجب علينا أن نقوم بإعداد قائمة self.trace لتخزين

المراجع في هذه الكائنات الرسومية، وأخيرا قائمة **self.trace** لتخزين المراجع في هذه الكائنات الرسومية، وأخيرا قائمة **self.controle** لتخزين مراجع لوحات التحكم الثلاثة.

- الأسطر من 13 إلى 15 : تمثيل ودجة العرض. لأن الصنف **OscilloGraphe()** تم اشتقاقه من صنف **Canvas()**، وهو دائما يمكنه تكوين هذا الودجة بإعادة تعريف الخيارات الخاصة لهذا الصنف (السطر 13).
- الأسطر من 18 إلى 20 : لتمثيل ثلاثة ودجات "لوحة تحكم"، نستخدم حلقة. ومراجعهم يتم حفظهم في قائمة **self.controle** التي تم تحضيرها في السطر 10. هذه لوحات التحكم يتم اشتقاقهم كعبيد من هذا الودجة، عن طريق البارمتر **self**. والبرامتر الثاني يقوم بتمرير اللون للمتحكم.
- السطران 23 و 24 : في وقت تمثيله، كل ودجة tkinter يتلقى تلقائيا سمة **master** التي تحتوي على مرجع النافذة الرئيسية للتطبيق. هذه السمة هي مفيدة بشكل خاص إذا تم إنشاء مثيل للنافذة الرئيسية بواسطة tkinter، كما هو الحال هنا.

تذكر أنه عندما نشغل تطبيق مثيل مباشر على ودجة مثل **Frame()**، على سبيل المثال (و هذا ما فعلناه في السطر 4)، tkinter يقوم بتمثيل تلقائيا نافذة الأصل لهذا الودجة (كائن من صنف **Tk()**).

مثل هذا الكائن الذي تم صنعه تلقائيا، ليس لدينا أي مرجع في الكود للوصول إليه، إذا كان من خلال هذه السمة الرئيسية التي قام tkinter بربطها تلقائيا لكل ودجة. ونحن نستخدم هذا المرجع لإعادة تحديد عنوان لافتة النافذة الرئيسية (في السطر 24)، وإرفاق معالج الأحداث (في السطر 23).

- الأسطر من 27 إلى 40 : الأسلوب الذي تم وصفه هنا هو معالج الأحداث <Maj-Ctrl-Z> الخاص بتسبب بودجاتنا **ChoixVibra()** (أو "معالج الأحداث")، في كل مرة يقوم فيها المستخدم بتنفيذ أي حركة على المتزلج أو في خانة الاختيار. وفي جميع الأحوال، قد تكون بعض الرسوم التي يجب حذفها أولا (السطر 28) بمساعدة الأسلوب **delete()** : الودجة **OscilloGraphe()** يرث هذا الأسلوب لهذا الصنف الأصل **Canvas()**.

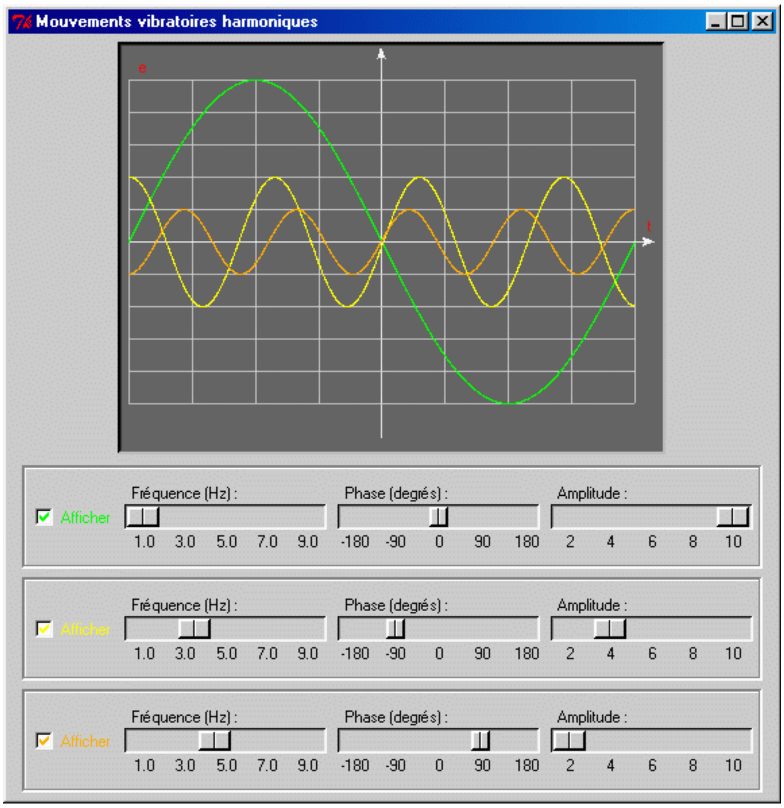
ثم، يتم رسم منحنيات جديدة، لكل من لوحات التحكم « coché la case Afficher ». كل واحد من الكائنات لديها في اللوحة رقم مرجعها، يتم إرجاعه عن طريق الأسلوب **traceCourbe()** للودجة الخاصة بنا **OscilloGraphe()**.

أرقام مراجع رسوماتنا يتم تخزينها في قائمة **self.trace**. وهو يسمح بحذف كل واحدة منها على حدة (انظر إلى التعليمة في السطر 28).

•الأسطر من 38 إلى 40 : قيم التذبذب والتردد والسعة يتم إرسالها إلى الأسلوب **traceCourbe()** هي سمات المثيل المقابل لكل واحد من لوحات التحكم الثلاثة، ويتم تخزينهم في قائمة **self.controle**. يمكننا إسترداد هذه السمات باستخدام صفات الأسماء بالنقاط .

تمارين

17.13 عدل السكريبت، بطريقة للحصول على الشكل بالأسفل (تعرض الشاشة مع شبكة المرجع، ولوحات التحكم المحاطة بالأخدود) :



18.13 عدل السكريبت، بحيث يتم إظهار 4 متحكمات رسومية بدل من 3، بالنسبة للون الرسم الرابع، أختر على سبيل المثال : أزرق، بني ...

19.13 الأسطر من 33 إلى 35، لقد حصلنا على قيم الطور والتردد والسعة التي تم إخيارها من قبل المستخدم بكل واحدة من ثلاثة لوحات التحكم، بالوصول المباشر لسمات المثيل المقابل. بيثون يسمح بهذا الإختصار (و هذا معناه تطبيقي) لكن هذه التقنية خطيرة. فهي تنتهك واحدة من التوصيات النظرية العامة "للبرمجة الشيئية"، التي توصي بأن الوصول إلى خصائص الكائن بطرق محددة. للإمتثال لهذه التوصية، أضف للصنف **ChoixVibra()** أسلوب التي تستدعي

valeurs()، والتي تقوم بإرجاع نفق (tuple) يحتوي على قيم الترددات والطور والسعة التي تم اختيارها.

الأسطر من 33 إلى 35 من هذا السكريبت يجب أن نستبدلها بهذا :

```
freq, phase, ampl = self.control[i].valeurs()
```

20.13 اكتب تطبيق صغير الذي يقوم بعرض نافذة مع لوحة وودجة متزلج (**Scale**). في اللوحة، قم برسم دائرة، والتي يمكن للمستخدم تغيير حجمها بمساعدة المتزحلق .

21.13 اكتب سكريت الذي يقوم بصنع صنفين : صنف التطبيق، الذي يشتق من **Frame()**، والذي يمثل المنشئ لوحة بحجم 400 × 400 بيكسل، بالإضافة لزران. في اللوحة، قم بعمل مثيل الكائن للصنف **Visage** الذي تم وصفه أدناه.

الصنف **Visage** يعرف الكائنات الرسومية لتمثيل وجوه أشخاص بسيطة. هذه الوجوه تتكون من دائرة رئيسية فيها ثلاثة دائراة بيضاوية الشكل والتي تمثل العينات والفم (مفتوح). الأسلوب (إغلاق) يسمح باستبدال الدائرة البيضاوية التي تمثل الفم بخط أفقي. والأسلوب (فتح) يسمح بإرجاع الفم على شكل الدائرة البيضاوية.

إثنين من الأزرار يتم تعريفها في صنف **Application** الذي ستسمح بفتح وغلق الفم للكائن **Visage** الموجود في اللوحة. يمكنك أن تتعلم من المثال في الصفحة93 لإقتباس جزء من الكود .

22.13 تكرين التركيب : تطوير قاموس ألوان.

الهدف : صنع برنامج صغير الذي يمكنه أن يساعدك بسرعة وسهولة على بناء قاموس ألوان جديد، التي ستسمح بالوصول إلى أي تقنية لون من خلال الاسم المستخدم بالفرنسية.

السياق : يجب أن نقوم بالتلاعب مع الكائنات الملونة مع tkinter، أنت تعرف أن هذه المكتبة الرسومية تقبل يشير إلى الألوان الأساسية في شكل سلاسل نصية تحتوي على الاسم باللغة الإنكليزية : red و blue و yellow و... إلخ.

أنت تعرف أن الحاسوب يمكنه معالجة البيانات الرقمية فقط. وهذا يعني عاجلا أم اجلا تعيين أي لون بترميزه إلى رقم. وينبغي بالطبع إعتماد اتفاقية لهذا، وتختلف من نظام لأخر. واحدة من هذه الإتفاقيات، وهي الكثير شيوعا، تمثيل اللون بثلاثة بيتات، والتي تشير إلى شدة اللون من مكوناته الثلاثة : الأحمر والأخضر والأزرق.

و يمكن استخدام هذه الاتفاقية مع tkinter للوصول إلى أي لون. يمكنك تععين اللون لأي عنصر رسومي، بمساعدة 7 رموز مثل `'#00FA4E'`. في هذه السلسلة، الرمز الأول (#) يعني أن القيمة بالنظام السداسي العشري. والستة الأرقام التي تليها تشير إلى 3 قيم سداسية عشرية لثلاثة مركبات من الأحمر والأخصر والأزرق.

لعرض المراسلات بين أي لون والكود، يمكنك إكتشاف برامج مختلفة لمعالجة الصور، مثل جيمب وإنكسكيب فهي برامج حرة ومفتوحة المصدر.

لأنه ليس من السهل علينا كبشر حفظ الكودات السادسية العشرية، لدى tkinter قاموس للتحويل، والذي يسمح باستخدام الأسماء الشائعة للعديد من الألوان الأكثر شيوعا، لكن هذا لا يعمل لأسماء الألوان باللغة الإنجليزية. الهدف نمن هذا التمرين هو سهولة صنع برنامج قاموس بالفرنسية، والتي يمكنك إدراجها بعد إذن في أي برنامج خاص بك. تبنيه مرة واحدة، هذا القاموس سيتكون شكله كالتالي :

{'vert':'#00FF00', 'bleu':'#0000FF', ... etc ...}.

المواصفات :

البرنامج الذي نريد تنفيذ هو برنامج رسومي، الذي يتمحور حول صنف. والتي سوف تشمل نافذة مع عدد من حقول الإدخال والأزرار، بحيث يمكن للمستخدم ترميز ألوان جديدة في كل مرة بكتابة اسمها باللغة الفرنسية في الحقل، وكود السداسي العشري في الحقل الأخر.

عندما يكون القاموس يحتوي على عدد من البيانات، فيكون من الممكن اختباره، وهذا معناه إدخال اسم اللون بالفرنسية ويقوم بإرجاع رمزه السداسي العشري بمساعدة الزر (مع عرض منطقة ملونة). سوف يتسبب زر في حفظ القاموس في ملف نصي. وزر آخر في إسترجاع القاموس من هذا الملف .

23.13 السكريبت بالأسفل هي أداة مسودة لمشروع رسم مجموعة من الأزهار على الشاشة بطرق مختلفة (و هذا المشروع قد يكون الخطوة الأولى لصنع لعبة). التمرين هو فحص هذا السكريبت وإكماله. وسوف يكون مكانك هو مواصلة العمل الذي قد بدأه شخص آخر، أو شخص طلب مساعدتك في المشاركة في العمل كفريق.

أ) أبدأ بفحص السكريبت وإضافة التعليقات، والتي هي في أسطر الملحوظة :: ***#، لإظهار أنك فهمت ما يجب القيام به في هذه الأماكن :

```
from tkinter import *
class FaceDom(object):
    def __init__(self, can, val, pos, taille =70):
        self.can =can
        # ***
        x, y, c = pos[0], pos[1], taille/2
        can.create_rectangle(x -c, y-c, x+c, y+c, fill ='ivory', width =2)
        d = taille/3
        # ***
        self.pList =[]
        # ***
        pDispo = [((0,0),), ((-d,d),(d,-d)), ((-d,-d), (0,0), (d,d))]
        disp = pDispo[val -1]
        # ***
        for p in disp:
            self.cercle(x +p[0], y +p[1], 5, 'red')

    def cercle(self, x, y, r, coul):
        # ***
        self.pList.append(self.can.create_oval(x-r, y-r, x+r, y+r, fill=coul))

    def effacer(self):
        # ***
```

```
        for p in self.pList:
            self.can.delete(p)

class Projet(Frame):
    def __init__(self, larg, haut):
        Frame.__init__(self)
        self.larg, self.haut = larg, haut
        self.can = Canvas(self, bg='dark green', width =larg, height =haut)
        self.can.pack(padx =5, pady =5)
        # ***
        bList = [("A", self.boutA), ("B", self.boutB),
                 ("C", self.boutC), ("D", self.boutD),
                 ("Quitter", self.boutQuit)]
        for b in bList:
            Button(self, text =b[0], command =b[1]).pack(side =LEFT)
        self.pack()

    def boutA(self):
        self.d3 = FaceDom(self.can, 3, (100,100), 50)

    def boutB(self):
        self.d2 = FaceDom(self.can, 2, (200,100), 80)

    def boutC(self):
        self.d1 = FaceDom(self.can, 1, (350,100), 110)

    def boutD(self):
        # ***
        self.d3.effacer()

    def boutQuit(self):
        self.master.destroy()

Projet(500, 300).mainloop()
```

ب) عدل هذا السكريبت، ليتناسب مع هذه المواصفات التالية : اللوحة يجب أن تكون حجمها أكبر : 600 × 600 بيكسل.

أزرار التحكم يجب نقلهم إلى اليمين.

حجم النقاط على الوجه يجب أن يكون يختلف نسبته لهذا الوجه .

الخيار 1 :

أبقي فقط زران A و B. مع كل إستخدان للزر A سوف يظهر 3 نردات جديدة (بنفس الطول، صغيرة نوعا ما) رتبت على عمود (عمودي)، القيم لهذه النردات يتم اختيارها عشوائيا ما بين 1 و 6. سيتم وضع كل عمود جديد على يمين سابقه. فإذا طلع 3 نردات وهي 1،2،4 (بأي ترتيب)، تظهر في النافذة (أو في اللوحة) رسالة "ربحت". سوف يقوم الزر B بحذف كل شيء (لكن ليس النقاط !) من جميع النردات المعروضة .

الخيار 2 :

أبقي فقط زران A و B. الزر A سوف يقوم بعرض 5 نردات متداخلة (هذا معناه مثل نقاط لوجه القيمة 5). القيم ستكون عشوائية بين 1 و 6، لكن لا يمكن أن تكون مكررة. الزر B يقوم بمسح كل شيء (لكن ليس النقاط) من جميع النردات المعروضة .

الخيار 3 :

أبقي فقط فقط 3 أزرار A و B و C. الزر A سيقوم بعرض 13 نرد من نفس الحجم مرتبة في شكل دائرة. كل استخدام للزر B يقوم بتغيير قيمة الزر الأول، ثم الثاني، ثم الثالث ... إلخ. القيمة الجديدة للنرد ستكون قيمة النرد السابق +1، إلا في حالات تكون فيها قيمة السابقة 6 : في هذه الحالة ستكون قيمة المتغير الجديد 1، وإلخ .... الزر C يقوم بمسح كل شيء (لكن ليس النقاط) من جميع النردات المعروضة .

الخيار 4 :

أبقي فقط فقط 3 أزرار A و B و C. الزر A سيقوم بعرض 12 نرد من نفس الحجم مرتبة في شكل سطرين من 6. قيم النردات ستكون في السطر الأول في ترتيب 1، 2، 3، 4، 5، 6. قيم النرد الثاني ستكون عشوائية بين 1 و 6. مع كل استخدام للزر B يقوم بتغيير قيمة نرد من السطر الثاني، وهذه القيمة ستبقى مختلفة بين نرد السطر الأول والسطر الثاني.

إذا كان النرد الأول في السطر الثاني تم الحصول على قيمة مقابله، سوف يتم تغيير القيمة النرد الثاني للسطر الثاني عشوائيا، وهكذا، إلى أن نحصل على 6 وجوه مشابهة للتي فوقها. الزر C يقوم بمسح كل شيء (لكن ليس النقاط) من جميع النردات المعروضة .

14

مع بعض الويدجات الإضافية

سوف نقدم هنا بعض الويدجات الجديدة، فضلا عن استخداماتها التي تعرفها. نحن لا نتظاهر في كل مرة أننا نبني مرجعًا ل *tkinter* : يمكنك العثور على المزيد على المواقع المخصصة. لك انتبه : ماوراء منظر الوثائق، هذه الصفحات مصممة لتعليمك حسب المثال كيفية صنع تطبيق بمساعدة الأصناف والكائنات. وسوف تكتشف بعض تقنيات بيثون التي لم نتناولها بعد، مثل تعبيرات *lambda* أو تحديد المهام الضمنية .

أزرار الراديو



وجدة "أزرار الراديو" يمكن أن يوفر للمستخدم مجموعة من الخيارات الخاصة التبادلية. التي يطلق عليها قياسا على أزرار الاختيار التي نجدها في الراديوات. تم تصميم هذه الأزرار بحيث تسمح باختيار خيار واحد في كل مرة : البقية ظهرت بشكل تلقائي.

الميزة الأساسية من هذه الودجات هي استخدامها دائما في مجموعات. جميع أزرار الراديو تنتمي إلى نفس المجموعة المرتبطة بواحد وهي أيضا مرتبط بمتغير tkinter، لكن كل واحدة تعين قيمة معينة.

عندما يقوم المستخدم باختيار أحد الأزرار، القيم المقابلة لهذا الزر يتم تعيينه لمتغير tkinter المشترك .

```
from tkinter import *

class RadioDemo(Frame):
    """Démo : utilisation de widgets 'boutons radio'"""
    def __init__(self, boss =None):
        """Création d'un champ d'entrée avec 4 boutons radio"""
        Frame.__init__(self)
        self.pack()
        # حقل إدخال يحتوي على نص صغير :
```

```
10#         self.texte = Entry(self, width =30, font ="Arial 14")
11#         self.texte.insert(END, "La programmation, c'est génial")
12#         self.texte.pack(padx =8, pady =8)
13#         # الاسم الفرنسي والاسم التقني للأنماط الأربعة للخطوط :
14#         stylePoliceFr =["Normal", "Gras", "Italique", "Gras/Italique"]
15#         stylePoliceTk =["normal", "bold", "italic"  , "bold italic"]
16#         # النمط الحالي يتم تخزينه في 'objet-variable' tkinter ;
17#         self.choixPolice = StringVar()
18#         self.choixPolice.set(stylePoliceTk[0])
19#         # صنع أربعة أزرار راديو' :
20#         for n in range(4):
21#             bout = Radiobutton(self,
22#                                text = stylePoliceFr[n],
23#                                variable = self.choixPolice,
24#                                value = stylePoliceTk[n],
25#                                command = self.changePolice)
26#             bout.pack(side =LEFT, padx =5)
27#
28#     def changePolice(self):
29#         """Remplacement du style de la police actuelle"""
30#         police = "Arial 15 " + self.choixPolice.get()
31#         self.texte.configure(font =police)
32#
33# if __name__ == '__main__':
34#     RadioDemo().mainloop()
```

تعليقات

- السطر 3 : هذا المرة أيضا، فضلنا بناء تطبيقنا الصغير كصنف مشتق من صنف **Frame()**، والذي يسمح لنا بالاندماج بسهولة في تطبيق أكثر أهمية.
- السطر 8 : عامة، نطبق أساليب تحديد مواقع الويدجات (**grid**، **pack()** أو **place()**) بعد تمثيلها، والذي يسمح لنا بحرية اختيار موقعه في داخل النافذة الأصل. وكما يظهر هنا، فمن الممكن التنبؤ بمواقعها في منشي الودجة.
- السطر 11 : ويدجات صنف الإدخال لديها العديد من الأساليب للوصول إلى السلسلة النصية المعروضة. الأسلوب **get()** يسمح باسترداد السلسلة بأكملها. الأسلوب **insert()** يسمح بإضافة حروف جديدة إلى أي مكان (و هذا يعني في البداية أو في النهاية أو حتى ضمن السلسلة الموجود إن وجدت). هذا الأسلوب يستخدم برامترين، الأول يشير إلى مكان الإضافة (استخدم **0** لإضافته إلى البداية، **END** لإضافته إلى نهاية السلسلة). الأسلوب **delete()** يسمح بمسح كل أو جزء من السلسلة. وهي تستخدم نفس البرامترات السابقة (انظر صفحة مشروع "رمز الألوان"، صفحة 205).
- السطران 14 و 15 : بدل من التمثيل في تعليمات منفصلة، نحن نفضل صنع 4 أزرار بمساعدة حلقة. الخيارات الخاصة لكل واحد منها يتم أولا إعدادها في قائمتين **stylePoliceFr** و **stylePoliceTk** : الأولى تحتوي على نصوص صغيرة التي يجب أن تظهر بجانب كل زر، والثانية تحتوي على قيم التي ينبغي أن ترتبط معها .

• السطران 17 و 18 : كما شرحنا في الصفحات السابقة، الأزرار الأربعة تشكل فريقا حول المتغير المشترك. هذا المتغير يأخذ القيمة المرتبطة مع زر الراديو الذي إختاره المستخدم. نحن لا نستطيع استخدام متغير مستقل لملء هذا الدور، لأن سمات كائنات tkinter الداخلية لا يمكن الوصول إليها إلا من أساليب مخصصة. مرة أخرى، نحن استخدمنا هنا كائن-متغير tkinter، من نوع سلسلة نصية، والذي قمنا بتمثيله من صنف **StringVar()**، والتي أعطيناها قيمة افتراضية في السطر 18.

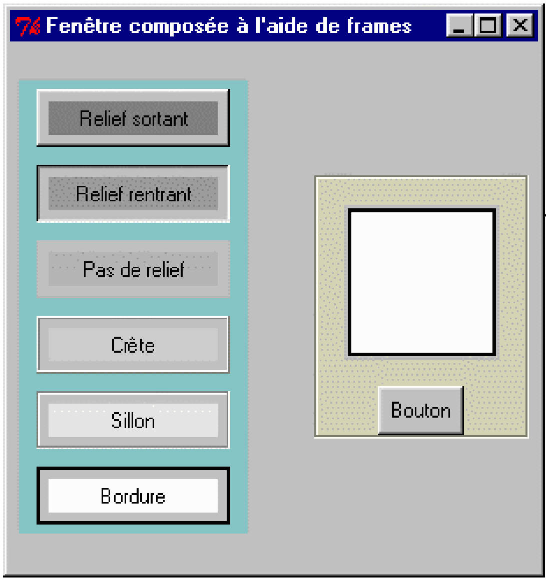
• الأسطر من 20 إلى 26 : نحن قمنا بتمثيل 4 أزرار راديو. كل واحد منه يتم تعيينه مع ملصق (Label) وقيمة مختلفة، لكن جميعهم مرتبطون بنفس متغير tkinter مشترك (**self.choixPolice**). لكنها تستدعي أسلوب **self.changePolice()**، في كل مرة يقوم فيها المستخدم بإجراء ضغطة من الفأرة على واحدة أو أخرى.

• الأسطر من 28 إلى 31 : يتم تغيير الخط بواسطة إعادة تكوين خيار الخط لودجة الإدخال. هذا الخيار ينتظر مصفوفة (tuple) تحتوي عى اسم الخط، وحجمه وربما أسلوبه. فإذا كان اسم الخط لا يحتوي على فراغات، مصفوفة (tuple) يمكنها أيضا استبداله بسلسلة نصية. على سبيل المثال :

```
('Arial', 12, 'italic')
('Helvetica', 10)
('Times New Roman', 12, 'bold italic')
"Verdana 14 bold"
"President 18 italic"
```

(انظر أيضا الأمثلة في صفحة 268).

استخدام الإطارات لتركيب نافذة



أنت تعرف بالفعل استخدام أصناف ودجات **Frame()** (إطار باللغة العربية)، بما في ذلك إنشاء ويدجات جديدة معقدة بالاشتقاق.

السكريبت الصغير بالأسفل يبين لك فائدة هذا الصنف لإعادة تجميع مجموعات من الويدجات وترتيبها بطريقة معينة داخل النافذة. وهذا يظهر لك أيضا استخدام خيارات زخرفية معينة(الحدود...إلخ).

لإنشاء نافذة على الجانب، استخدمنا إطارين **f1** و **f2** بطريقة لصنع مجموعتين من الويدجات، واحدة على اليمين والأخرى على اليسار. ولقد لونا هذين الإطارين لتسليط الضوء عليهم بشكل جديد، ولكن هذا ليس ضروريا.

الإطار **f1** يحتوي على 6 إطارات أخرى، والتي تحتوي كل واحدة منها على ودجة من صنف **Label()**. الإطار **f2** يحتوي على ودجة **Canvas()** وودجة **Button()**. الألوان والتقليم هي خيارات بسيطة .

```
 1#  from tkinter import *
 2#
 3#  fen = Tk()
 4#  fen.title("Fenêtre composée à l'aide de frames")
 5#  fen.geometry("300x300")
 6#
 7#  f1 = Frame(fen, bg = '#80c0c0')
 8#  f1.pack(side =LEFT, padx =5)
 9#
10#  fint = [0]*6
11#  for (n, col, rel, txt) in [(0, 'grey50', RAISED, 'Relief sortant'),
12#                             (1, 'grey60', SUNKEN, 'Relief rentrant'),
13#                             (2, 'grey70', FLAT, 'Pas de relief'),
14#                             (3, 'grey80', RIDGE, 'Crête'),
15#                             (4, 'grey90', GROOVE, 'Sillon'),
16#                             (5, 'grey100', SOLID, 'Bordure')]:
17#      fint[n] = Frame(f1, bd =2, relief =rel)
18#      e = Label(fint[n], text =txt, width =15, bg =col)
19#      e.pack(side =LEFT, padx =5, pady =5)
20#      fint[n].pack(side =TOP, padx =10, pady =5)
21#
22#  f2 = Frame(fen, bg ='#d0d0b0', bd =2, relief =GROOVE)
23#  f2.pack(side =RIGHT, padx =5)
24#
25#  can = Canvas(f2, width =80, height =80, bg ='white', bd =2, relief =SOLID)
26#  can.pack(padx =15, pady =15)
27#  bou =Button(f2, text='Bouton')
28#  bou.pack()
29#
30#  fen.mainloop()
```

تعليقات

- الأسطر من 3 إلى 6 : لتبسيط المظهر إلى أقصى حد ممكن، لم نبرمج هذا المثال كصنف جديد. لاحظ في السطر 5 فائدة الأسلوب **geometry()** لتعيين حجم النافذة الرئيسية.
- السطر 7 : تمثيل الإطار الأيسر. يتم تحديد لون الخلفية (لون أزرق فاتح) عن طريق البرامتر **bg** (الخلفية - background). هذه السلسلة النصية تحتوي وصف سداسي عشري لثلاثة مركبات الأحمر والأخضر والأزرق من اللون المطلوب. بعد الرمز **#** لإشارة إلى أن القيمة الرقمية هي بالنظام السداسي العشري، وهنالك ثلاثة مجموعات مع الأرقام والحروف. كل واحد من هذه المجموعات يحتوي على رقم ما بين 1 و 255. 80 يقابل 128 و c0 يقابل 192 بالنظام العشري. في مثالنا، المركبات الأحمر والأخضر والأزرق التي تحتاجها لتمثيل اللون هي على التوالي 128 و 192 و 192 . padx =5

- لنطبق هذه التقنية، يتم الحصول على الأسود مع ‎#000000‎ـ والأبيض مع **‎#ffffff‎**ـ والأحمر مع **‎#ff0000‎**ـ والأزرق الداكن مع ‎#000050‎، إلخ.
- السطر 8 : بما أننا طبقنا الأسلوب **pack()**ـ سيتم تغيير إطار تلقائيا حسب محتوياته. الخيار **side =LEFT** ليضع المحتويات على يسار النافذة الرئيسية. والخيار **padx =5** يضع مساحة 5 بيكسلات على اليمين واليسار (فارغة) (و يمكننا ترجم "padx" بالتباعد الأفقي) .
- السطر 10 : في إطار **f1** الذي نقوم بإعداده، نحن ستقنو بجمع 5 إطارات متشابهة تحتوي على واحدة منها على ملصق. الكود المقابل سيكون أكثر بساطة وأكثر فعالية إذا قمنا بتمثيل هذه الويدجات في قائمة بدل من متغيرات مستقلة. نحن نعد الآن قائمة مع 6 عناصر وسوف نستبدلها لاحقا.
- الأسطر من 11 إلى 16 : لبناء 6 إطارات متشابهة، سوف نقوم باستعراض قائمة من 6 مصفوفات مغلقة تحتوي على الخاصيات التي ينفرد بها كل إطار. كب واحد من هذه المصفوفات تتكون من 4 عناصر : مؤشر وثابت tkinter يعرف نوع التخفيف، سلسلتين نصيتين تصف اللون والكتابة في الملصق.
- الحلقة **for** يتم تنفيذها 6 مرت على 6 عناصر من القائمة. مع كل تكرار، محتوى إحدى المصفوفات المغلقة يتم تعيينه للمتغيرات **rel**، **col**، **n** و **txt** (ثم يتم تنفيذ تعليمات الأسطر من 17 إلى 20) .

> تدوير قائمة من المصفوفات المغلقة باستخدام حلقة *for* والذي هو بناء مدمج خاص. يسمح بأداء العديد من المهام مع عدد قليل جدا من التعليمات .

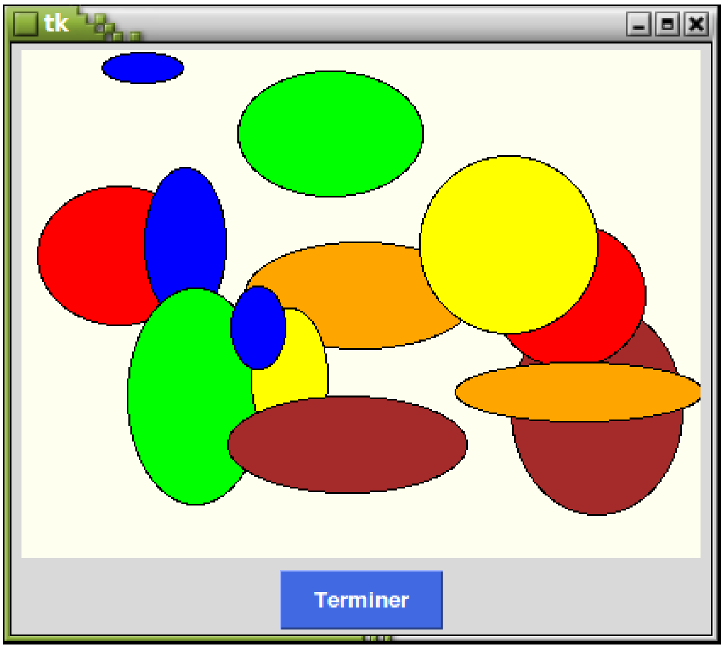
- السطر 17 : الإطارات 6 يتم تمثيلهم كعناصر للقائمة **fint**. وكل واحدة تم تزيينها بحدود من 2 بيكسل واسع مع وجود تأثير التخفيف.
- الأسطر من 18 إلى 20 : الملصقات جميعها بنفس الحجم، لكن النصوص التي بداخلها وألوانها هي المختلفة. وقد استخدمنا الأسلوب **pack()**ـ لحجم الملصق التي يتم تحديدها بإطارات صغير. والتي بدورها تحدد طول الإطار الذي يتم تجميعه (الإطار f1). الخيارات **padx** و **pady** يسمحون بحجز مساحة صغيرة حول كل ملصق، ومساحة صغيرة حول كل إطار صغير. الخيار **side =TOP** يضع كل الإطارات الست والصغير واحد تحت الأخر في الإطار الذي يحتويهم f1.
- الأسطر 22 و 23 : إعداد الإطار **f2** (الإطار الأيمن). لونه سيكون أصفر، وسنقوم بإحاطة حدود حول زخرفة الأخدود.
- الأسطر من 25 إلى 28 : الإطار: **f2** يحتوي على لوحة وزر. لاحظ مرة أخرى أن استخدام الخيارات **padx** و **pady** لترك مساحة حول الويدجات (على سبيل المثال، انظر لحالة أحد الأزار، التي لم تستخدم هذا الخيار، وبالتالي فهو يأتي مع حافة الإطار المحيطة به). كما فعلنا للإطارين السابقين، وضعنا حافة حول اللوحة. اعلم أن الويدجات الأخرى تقبل هذا النوع من الزخرفة مثل : الأزرار وحقول الإدخال ...إلخ .

كيفية نقل رسومات بمساعدة الفأرة

الودجة اللوحة هي أحد نقاط القوة لمكتبة tkinter الرسومية. فهي تشتمل على مجموعة كبيرة جدا من الأدوات الفعالة التي تعالج الرسوم. السكريبت أدناه يظهر لك بعض التقنيات الأساسية. فإذا أردت معرفة المزيد، خاصة فيما يتعلق بمعالجة الرسومات التي تتكون من عدة أجزاء، يرجى الرجوع إلى كتاب مرجعي للتعامل مع tkinter.

في بداية تطبيقنا الصغير، مجموعة من الرسومات المرسومة عشوائيا على اللوحة (و هي مجموعة من الدوائر البيضاوية الملونة). ويمكنك نقل أي واحدة من هذه الرسومات من خلال فأرتك.

عندما يتم نقل إحدى الرسومات يتم تمريرها إلى الامام مقارنة بالآخرين، وحدودها نصبح أكثر سماكة طوال وقت التعامل معها.



لفهم هذه التقنية المستخدمة، يجب أن تتذكر برنامجا يستخدم الواجهة الرسومية (يتم التحكم به بواسطة الأحداث)، (أعد النظر إذا أردت للتفسيرات في صفحة88). في هذا التطبيق، سوف نقوم بوضع تقنية تتفاعل مع الأحداث : "ضغط على الزر الأيمن للفأرة"، "تحريك الفأرة والزر الأيمن لا يزال مضغوط"، "تحرير الزر الأيمن".

يتم إنشاء هذه الأحداث من قبل نظام التشغيل وبدعم من واجهة tkinter. عملنا البرمجي هو ربطها بمعالجات مختلفة (دالات أو أساليب).

لتطوير هذا التطبيق الصغير بعد "فلسفة الكائن"، ونحن نفضل صنع صنف جديد **Bac_a_sable**، الذي يشتق من اللوحة الأساسية، ويتم إدراج الوظائف المطلوبة، بدلا من برمجة هذه الوظائف في مستوى البرنامج الرئيسي، سوف نبنيها على لوحة عادية. وبالتالي ننتج كود قابل لإعادة الاستخدام .

```
from tkinter import *
from random import randrange

class Bac_a_sable(Canvas):
    "Canevas modifié pour prendre en compte quelques actions de la souris"
    def __init__(self, boss, width=80, height=80, bg="white"):
        # استدعاء منشئ الصنف الأصل :
        Canvas.__init__(self, boss, width=width, height=height, bg=bg)
        # ربط الأحداث " الفأرة" بهذه الوحدة :
        self.bind("<Button-1>", self.mouseDown)
        self.bind("<Button1-Motion>", self.mouseMove)
        self.bind("<Button1-ButtonRelease>", self.mouseUp)

    def mouseDown(self, event):
```

```
        "Opération à effectuer quand le bouton gauche de la souris est enfoncé"
        self.currObject =None
        # event.x وevent.y تحتوي على إحداثيات نقرة الفأرة :
        self.x1, self.y1 = event.x, event.y
        # <find_closest> تقوم بإرجاع مرجع الرسم الأقرب :
        self.selObject = self.find_closest(self.x1, self.y1)
        # تغيير سمك محيط الرسم :
        self.itemconfig(self.selObject, width =3)
        # <lift> تمرير الرسم إلى المقدمة :
        self.lift(self.selObject)

    def mouseMove(self, event):
        "Op. à effectuer quand la souris se déplace, bouton gauche enfoncé"
        x2, y2 = event.x, event.y
        dx, dy = x2 -self.x1, y2 -self.y1
        if self.selObject:
            self.move(self.selObject, dx, dy)
            self.x1, self.y1 = x2, y2

    def mouseUp(self, event):
        "Op. à effectuer quand le bouton gauche de la souris est relâché"
        if self.selObject:
            self.itemconfig(self.selObject, width =1)
            self.selObject =None

if __name__ == '__main__':      # ---- Programme de test ----
    couleurs =('red','orange','yellow','green','cyan','blue','violet','purple')
    fen =Tk()
    # وضع في اللوحة - رسم من 15 شكل بيضوي ملون :
    bac =Bac_a_sable(fen, width =400, height =300, bg ='ivory')
    bac.pack(padx =5, pady =3)
    # زر الخروج :
    b_fin = Button(fen, text ='Terminer', bg ='royal blue', fg ='white',
                   font =('Helvetica', 10, 'bold'), command =fen.quit)
    b_fin.pack(pady =2)
    # رسم 15 شكل بيضوي مع إحداثيات ولون عشوائيان :
    for i in range(15):
        coul =couleurs[randrange(8)]
        x1, y1 = randrange(300), randrange(200)
        x2, y2 = x1 + randrange(10, 150), y1 + randrange(10, 150)
        bac.create_oval(x1, y1, x2, y2, fill =coul)
    fen.mainloop()
```

تعليقات

السكريبت يحتوي على أساسيات تعريف صنف رسومي مشتق من **Canvas()**.

هذا الصنف الجديد من المرجح أن يتم إعادة استخدامه في مشاريع أخرى، وبالتالي اختبار هذه الصنف في البنية الكلاسيكية : `if __name__ ="__main"`:.و بالتالي يمكن استخدام هذا السكريبت كما هو، ووحدة لاستدعائها لتطبيقات أخرى.

منشئ ودجة الخاص بنا الجديد **Bac_a_sable()** ينتظر مرجع لودجة الأصل (boss) كبرامتر أول، كما في الاتفاقية المعتادة. ويستدعي منشئ الصنف الأصل، ثم ينفذ الآلية المحلية.

فـــي هـــذه الحالـــة، فـــإنه تـــم ربـــط ثلاثـــة معرفـــات أحـــداث <Button1-Motion>، <Button-1> و<Button1-ButtonRelease> مع أسماء ثلاثة أساليب مختارة كمعالجة لهذه الأحداث[74].

عندما يضغط المستخدم على الزر الأيمن للفأرة، فإنه يتم تفعيل الأسلوب **mouseDown()** ويقوم نظام التشغيل بتمرير في البرامتر كائن **event** الذي يحتوي على سمات **x** ، **y** والتي هي إحداثيات المؤشر في موقع اللوحة، الذي حدد في وقت الضغطة.

نحن نقوم بتخزين مباشرة هذه الإحداثيات في متغير مثل **self.x1** و **self.x2** لأننا في حاجة إليها في مكان آخر. ثم قمنا باستخدام الأسلوب **find_closest()** لودجة اللوحة، التي قمنا بإرجاع مرجع رسم الأقرب. هذا الأسلوب يقوم دائما بإرجاع مرجع، حتى لو كانت ضغطة الزر ليست داخل رسم.

البقية سهلة الفهم : مرجع الرسم تم حفظه في متغير مثيل، ويمكننا استخدام أساليب أخرى للوحة الأساسية لتعديل خصائصه. وفي هذه الحالة، نحن نستخدم الأسلوب **itemconfig()** و **lift()** لزيادة رشاقة المخطط ونمررة للمقدمة.

يتم استدعاء "ناقل" الرسم من خلال الأسلوب

يتم استدعاء "ناقل" الرسم من خلال الأسلوب **mouseMove()** والذي يتم استدعاؤه في كل مرة يتم فيها تحريك الفأرة والزر الأيسر مضغوط. الكائن **event** يحتوي هذه المرة أيضا على إحداثيات لمؤشر الفأرة، في نهاية النقل. ونحن نستخدمها لحساب الفرق بين الإحداثيات الجديدة والقديمة، من أجل تمرير الأسلوب **move()** لودجة اللوحة، والتي سوف تنقل نفسها.

ونحن لا يمكننا استدعاء هذا الأسلوب إذا كان هنالك تنفيذ كائن موجود (هذا دور متغير مثيل **selObject**)، ونحن نتأكد أيضا من أن الإحداثيات الجديدة المكتسبة تم حفظها.

الأسلوب **mouseUp()** تنهي العمل. عندما يتم نقل الرسم إلى وجهته، يبقى فقط إلغاء تحديد والانتقال إلى سمك الأولي.

و هذا لا يمكن أن يعتبر إذا كان هنالك بالفعل اختيار، بطبيعة الحال.

في جسم البرنامج الاختبار، قمنا بتمثيل 15 رسم دون القلق حول حفظ مراجعه في متغيرات. يمكنك فعل هذا لأن tkinter تقوم بحفظ مرجع داخل لكل من هذه الكائنات (انظر صفحة 104)

> ملاحظة : إذا كنت تعمل مع مكتبات رسومية أخرى، فربما يجب عليك أن توفر ذاكرة من هذه المراجع .

---

74 تذكير : معالج الأحداث لا ينقل الرسائل، إلا إذا وقعت أحداث محددة في اللوحة . نقرات الفأرة المضغوطة خارجا لا تقوم بأي تأثير.

الرسومات هي مجرد أشكال بيضوية ملونة. يتم إختيار ألوانها عشوائيا في قائمة من 8 إحتمالات، و يتم تحديد مؤشر اللون عن طريق الدالة **randrange()** التي تم إستدعائها من وحدة **random**.

ويدجات مكملة، ويدجات مركبة

إذا إستكشفت الوثائق الكبيرة الموجود على الإنترنت لـ tkinter، فسوف تعرف أنه يوجد توسعات ملحقة ، في شكل مكتبات. هذه الملحقات تقدم أصناف ودجة إضافية يمكن اعتبارها لا تقدر بثمن لتطوير سريع للتطبيقات المعقدة. لا يمكننا بالطبع أن نتحدث عن كل هذه الويدجات في هذه الدورة. فإذا كنت مهتما، حاول زيارة مواقع ذات صلة لمكتبات **Tix** و **Ttk** (و الأخريات). مكتبة **Tix** تقترح عليك أكثر من 40 ودجة إضافية. مكتبة est **Ttk** تهدف إلى "تلبيس" الويدجات مع ثيمات مختلفة (ستيل الأزرار والنوافذ ...إلخ). وكتبت بعض هذه المكتبات ببيثون، مثل **Pmw** (*Python Mega Widgets*).

و مع ذلك، يمكنك القيام بالكثير من الأشياء دون البحث عن موارد أخرى لمكتبة tkinter القياسية. في الواقع يمكنك بنفسك بناء أصناف ويدجات جديدة مركبة وفقا لاحتياجاتك. وهذا قد يستغرق بعض العمل في البداية، ولكن عندما تقوم بذلك، فإنك تتحكم بدقة ما في تطبيقك الخاص، وتضمن قابلية المحمولية لجميع الأنظمة التي تقبل بيثون، لأن tkinter توزع كجزء من بيثون القياسية. في الحقيقة، عندما تستخدم مكتبات طرف ثالث، يجب عليك دائما التحقق من توافرها وتوافقها للآلات التي تستهدفها برامجك، وتثبيتها، إذا لزم الأمر.

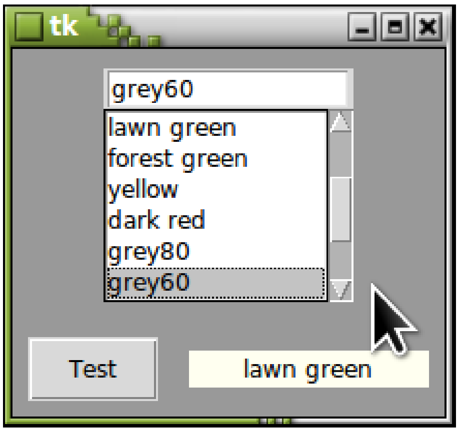
الصفحات التالية تشرح المبادئ العامة التي يتعين تنفيذها بنفسك للأصناف ودجة المركبة، مع بعض الأمثلة الأكثر فائدة .

مكعبات كومبو مبسطة

التطبيق الصغير أدناه يوضح لك كيفية بناء صنف جديد من ودجة من نوع مكعب كومبو. ويعرف أنه ودجة يربط بين حقل الإدخال وعلبة القائمة : يمكن للمستخدم الدخول إلى النظام والذي هو عناصر القائمة (بالضغط على اسمها)، أو إذا كان العنصر غير موجود (يكتب اسم جديد من خلال حقل الإدخال). نحن بسطنا المشكلة فقط من خلال ترك اللائحة واضحة دئما، لكن من الممكن تحسين هذا الودجة بحيث يأخذ الشكل التقليدي لحقل الإدخال يرافقه زر صغير يتسبب في ظهور قائمة، ويتم إخفاء هذا في البداية (أنظر للتمرين 14.1 صفحة 274).

و كما نتصور، ودجة الخاصة بنا combo سيتم تجميعها في كيان واحد من 3 ويدجات tkinter أساسية : حقل إدخال، مربع قائمة (listbox) وشريط تمرير.

مربع القائمة وشريط التمرير سيتم ربطهم، لأن شريط التمرير يسمح بتجاوز قائمة علبته. وسيتم أيضا التأكد من أن شريط التمرير سيكون دائما في نفس ارتفاع العلبة، بصرف النظر عن الحجم الذي اخترته لذلك .



سوف نضع علبة قائمة وشريط تمرير جنبا إلى جنب في إطار (Frame)، ووضعه مع محتوياته في أسفل حقل الإدخال، في إطار آخر عام أكثر. وسوف تكون جميع الويدجات التي لدينا مركبة.

لاختبار الودجة الخاص بنا، سوف نقوم بتضمين تطبيق بسيط جدا : عندما يقون المستخدم باختيار لون القائمة (و يمكن أيضا إدخال اسم اللون مباشرة في حقل الإدخال)، وهذا سيتسبب تلقائيا بتغير لون خلفية النافذة الرئيسية.

في هذه نافذة الأساسية، سوف نضيف ملصق وزر، لنظهر لك كيف يمكنك الوصول إلى الاختيار الذي تم اختياره في مكعب كومبو نفسها (الزر سيقوم بعرض اسم آخر لون تم اختياره) .

```
from tkinter import *

class ComboBox(Frame):
    "Widget composite associant un champ d'entrée avec une boîte de liste"
    def __init__(self, boss, item='', items=[], command ='', width =10,
                 listSize =5):
        Frame.__init__(self, boss)    # منشئ الصنف الأصل
                                      # (هو مرجع الودجة السيد <boss>)
        self.items =items             # عناصر لوضعها في علبة القائمة
        self.command =command         # دالة لاستدعائها عند النقر أو ضغط زر الإدخال
        self.item =item               # عناصر مدخلات أو محددة

        # حقل الإدخال :
        self.entree =Entry(self, width =width)          # العرض بالحروف
        self.entree.insert(END, item)
        self.entree.bind("<Return>", self.sortieE)
        self.entree.pack(side =TOP)

        # علبة القائمة, شريط تمرير :
        cadreLB =Frame(self)                       # إطار لجمع الودجتين
        self.bListe =Listbox(cadreLB, height =listSize, width =width-1)
        scrol =Scrollbar(cadreLB, command =self.bListe.yview)
        self.bListe.config(yscrollcommand =scrol.set)
        self.bListe.bind("<ButtonRelease-1>", self.sortieL)
        self.bListe.pack(side =LEFT)
        scrol.pack(expand =YES, fill =Y)
        cadreLB.pack()
```

```
28#
29#         # تعبئة علبة القائمة مع العناصر :
30#         for it in items:
31#             self.bListe.insert(END, it)
32#
33#     def sortieL(self, event =None):
34#         # استخراج قائمة العناصر المحددة :
35#         index =self.bListe.curselection()      # إرجاع مصفوفة مغلقة للمؤشر
36#         ind0 =int(index[0])                    # نبقي الأول فقط
37#         self.item =self.items[ind0]
38#         # تحديث حقل الإدخال مع العنصر المختار :
39#         self.entree.delete(0, END)
40#         self.entree.insert(END, self.item)
41#         # تشغيل الأمر المحدد مع العنصر المختار كبرامتر :
42#         self.command(self.item)
43#
44#     def sortieE(self, event =None):
45#         # تشغيل الأمر المحدد مع برامتر تم ترميزه على هذا النحو :
46#         self.command(self.entree.get())
47#
48#     def get(self):
49#         إرجاع العنصر الأخير المحدد في علبة القائمة
50#         return self.item
51#
52# if __name__ =="__main__":              # --- برنامج التجربة ---
53#     def changeCoul(col):
54#         fen.configure(background = col)
55#
56#     def changeLabel():
57#         lab.configure(text = combo.get())
58#
59#     couleurs = ('navy', 'royal blue', 'steelblue1', 'cadet blue',
60#                 'lawn green', 'forest green', 'yellow', 'dark red',
61#                 'grey80','grey60', 'grey40', 'grey20', 'pink')
62#     fen =Tk()
63#     combo =ComboBox(fen, item ="néant", items =couleurs, command =changeCoul,
64#                     width =15, listSize =6)
65#     combo.grid(row =1, columnspan =2, padx =10, pady =10)
66#     bou = Button(fen, text ="Test", command =changeLabel)
67#     bou.grid(row =2, column =0, padx =8, pady =8)
68#     lab = Label(fen, text ="Bonjour", bg ="ivory", width =15)
69#     lab.grid(row =2, column =1, padx =8)
70#     fen.mainloop()
```

تعليقات

• الأسطر من 5 إلى 8 : منشئ الودجة الخاص بنا ينتظر مرجع الودجة الأصل (boss) كبرامتر أول، ثم بقية الاتفاقية المعتادة. البرامترات الأخرى تسمح بتوفير نص افتراضي في حقل الإدخال (item)، ويتم إضافة قائمة العناصر إلى علبة (items)، وتعيين دالة لاستدعائها عندما يقوم المستخدم بالضغط في القائمة، أو من لوحة المفاتيح على زر الإدخال (command). أبقينا الأسماء الإنجليزية لهذه البرامترات، بحيث يمكن لودجتنا استخدامه مع نفس اتفاقيات الويدجات الأساسية التي يشتق منها.

- الأسطر 15و 39 و 40 : أساليب ودجة الإدخال تم وصفها سابقا (انظر للصفحة235). تذكر ببساطة الأسلوب **insert()** الذي يسمح بإضافة نص إلى حقل الإدخال، دون إزالة النص السابق. البرامتر الأول يسمح بتحديد مكان الإدراج النص الحالي لتأخذ مكانها. ويمكن أن يكون ذا عدد صحيح أو قيمة رمزية (عن طريق الاستدعاء لكامل لوحدة tkinter في السطر 1، سوف نقوم بجلب مجموعة من المتغيرات العامة، مثل **END**، التي تحتوي على قيم الرمزية و **END** تشير إلى نهاية النص السابق).
- الأسطر 16 و 24 : ويرتبط الحدثان مع أساليب محلية : يتم تحرير الزر الأيمن للفأرة إذا كان مؤشره موجودا في علبة القائمة (الحدث **<ButtonRelease-1>**) وإذا ضغطنا على زر الإدخال (الحدث **<Return>**).
- السطر 21 : إنشاء علبة القائمة (صنف الأصل **Listbox**). ويعرف حجمه بعدد من الأحرف في السطر الحالي. ونقوم بطرح واحد أو اثنين، لتعويض المكان الذي يشغله شريط التمرير (مجموعة من اثنين لديها ما يقرب من نفس عرض حقل الإدخال).
- السطر 22 : إنشاء شريط التمرير العمودي (صنف الأساس **Scrollbar**). الأمر الذي سيرتبط به هو **command =self.bListe.yview** يشير لأسلوب الودجة Listbox التي سيتم استدعاؤها والتي ستتسبب في انتقال قائمة في داخل العلبة، عندما نقوم بتحريك شريط التمرير .
- بشكل متناظر، يجب علينا إعادة تكوين علبة القائمة للإشارة إلى أسلوب الودجة Scrollbar الذي تم استدعاؤه، بحيث يعبر موقع شريط التمرير بشكل صحيح على موقع للعنصر الذي تم اختياره في القائمة. وليس من الممكن الإشارة إلى هذا الأمر في سطر تعليمات إنشاء صندوق القوائم، في السطر 21، لأن الودجة Scrollbar -في هذه اللحظة- لم يوجد بعد. لذا تم إبعاد المرجع.[75]
- السطر 33 : هذا الأسلوب يتم استدعاؤه في كل مرة يقوم فيها المستخدم بتحديد عنصر في القائمة. فهي تقوم باستدعاء الأسلوب **curselection()** لودجة Listbox الأساس. وهذا يقوم بإرجاع مصفوفة مغلقة للمؤشرات، لأن هذا مقصود من مطوري tkinter التي تمكن المستخدم تحديد العديد من الاعناصر في القائمة (بمساعدة الزر <Ctrl>). ومع ذلك نحن نفترض أن وحدا كان لافتا، وبالتالي استرداد عنصر واحد فقط من هذه المصفوفة المغلقة. في السطر 47، يمكننا إذا إستخراج عنصر المقاب من القائمة واستخدامه، في كل تحديث لحقل الإدخال (لاسطر 42) وكذلك مرجع تمثيل الودجة (في حالة التطبيق الصغير لدينا، سيكون دالة **changeCoul()**).

---

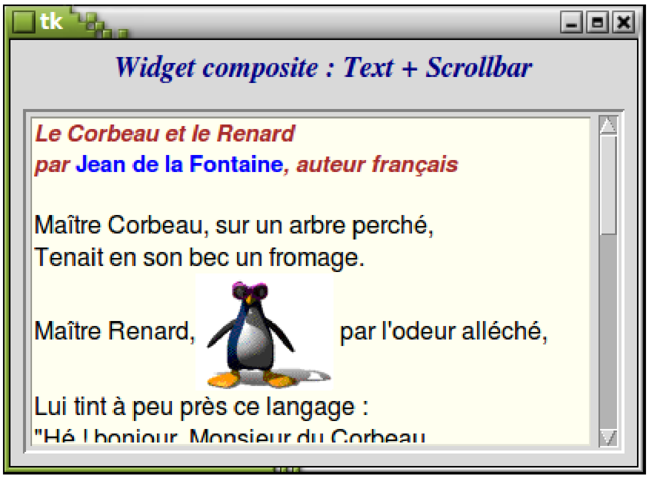
[75] Nous aurions pu aussi faire exactement l'inverse, c'est-à-dire créer d'abord l'ascenseur sans indication de commande, puis créer la boîte de liste en indiquant la commande d'accès à l'ascenseur dans la ligne d'instanciation, et enfin reconfigurer l'ascenseur en lui indiquant la commande de défilement de la liste :

```
scrol =Scrollbar(cadreLB)
self.bListe =Listbox(cadreLB, height =listSize, width =width-1,
                     yscrollcommand =scrol.set)
scrol.config(command =self.bListe.yview)
```

- الأسطر من 44 إلى 46 : يتم استدعاء نفس الأمر عندما يقوم المستخدم بتفعيل زر الإدخال بعد ترميز سلسلة نصية في حقل الإدخال. البرامتر **event**، لا يتم استخدامه هنا، وهو يسمح باستعادة العنصر أو العناصر المرتبطة .
- السطران 48 و 49 : ولقد قمنا أيضا بتضمين الأسلوب **get()**ـ بعد الاتفاقية التالية للويدجات الأخرى، للسماح بحرية باستعادة آخر عنصر تم تحديده .

الودجة نص يرافقه شريط تمرير

سوف نقوم بالشروع بنفس طريقة المثال السابق، سوف تقوم بربط الويدجات القياسية لـ tkinter بطرق متعددة. وبالتالي نحن نقدم أدناه ودجة مركب يمكن استخدامه لرسم نظام معالجة نصوص بدائي. عنصره الرئيسي هو ودجة النص القياسي الذي يمكنه عرض نصوص منسقة، هذا معناه دمج مختلف سمات الستايل (مثل تغميق، ومائل ...)مع النصوص، وكذلك الخطوط المختلفة، واللون وحتى الصور. سوف نقوم ببساطة بربط مع الشريط التمرير العمودي لنظهر لكم، مرة أخرى، التمثيل الذي تستطيع صنعه بين هذه المركبات.



اليدجة نص قادر على تفسير نظام "العلامات" المدرجة في أي مكان في النص. معهم، يمكنك وضع معايير، عمل وصلة، وجعل العناصر قابلة للعرض (نصوص أو صور)، بطريقة يمكن استخدامه لتشغيل مجموعة متنوعة من الأليات.

على سبيل المثال، في التطبيق الموصوف أدناه، عندما نقوم بالضغط على اسم "Jean de la Fontaine" بمساعدة الزر الأيمن للفأرة، سيتسبب تلقائيا في تمرير تلقائي للنص (تمرير)، حتى يصبح وصف هذا الكاتب مرئيا في الودجة (انظر للسكريبت المقابل في الصفحة التالية). ومن الميزات الأخرى الموجودة، يمكن بمساعدة الفأرة تحديد أي جزء من النص المعروض لتعديله، لكن نحن هنا نقدم الأساسيات. يرجى الرجوع إلى كتاب أو مواقع متخصصة لمزيد من المعلومات .

إدارة النص المعروض

يمكنك الوصول إلى أي جزء من نص يدعم ودجة النص مع اثنين من المفاهيم التكميلية، الفهارس والمؤشرات :

- تتم الإشارة إلى كل حرف من النص المعروض بمؤشر، الذي يجب أن يكون سلسلة نصة تحتوي على قيمتين متصلتين بنقطة (على سبيل المثال : "5.2"). هاتان القيمتان على التوالي تشيران إلى رقم السطر ورقم العمود أين يوجد الحرف.
- يمكن ربط أي جزء من النص مع واحدة أو أكثر من العلامات، التي أنت حر في اختيار اسمها وخصائصها. وهذه تسمح لك بتعريف الخط والألوان ولون الخلفية والعناصر المرتبطة وإلخ ...

> لفهم النص أدناه، يرجى أن تعرف أنه سيتم التعامل مع ملف يسمى ***CorbRenard.txt*** تم ترميزه بواسطة *latin-1* .

```
from tkinter import *

class ScrolledText(Frame):
    """Widget composite, associant un widget Text et une barre de défilement"""
    def __init__(self, boss, baseFont ="Times", width =50, height =25):
        Frame.__init__(self, boss, bd =2, relief =SUNKEN)
        self.text =Text(self, font =baseFont, bg ='ivory', bd =1,
                        width =width, height =height)
        scroll =Scrollbar(self, bd =1, command =self.text.yview)
        self.text.configure(yscrollcommand =scroll.set)
        self.text.pack(side =LEFT, expand =YES, fill =BOTH, padx =2, pady =2)
        scroll.pack(side =RIGHT, expand =NO, fill =Y, padx =2, pady =2)

    def importFichier(self, fichier, encodage ="Utf8"):
        "insertion d'un texte dans le widget, à partir d'un fichier"
        of =open(fichier, "r", encoding =encodage)
        lignes =of.readlines()
        of.close()
        for li in lignes:
            self.text.insert(END, li)

def chercheCible(event=None):
    "défilement du texte jusqu'à la balise <cible>, grâce à la méthode see()"
    index = st.text.tag_nextrange('cible', '0.0', END)
    st.text.see(index[0])

### البرنامج الرئيسي : نافذة مع ملصق و 'ScrolledText' ###
fen =Tk()
lib =Label(fen, text ="Widget composite : Text + Scrollbar",
           font ="Times 14 bold italic", fg ="navy")
lib.pack(padx =10, pady =4)
st =ScrolledText(fen, baseFont="Helvetica 12 normal", width =40, height =10)
st.pack(expand =YES, fill =BOTH, padx =8, pady =8)

# تعريف العلامات, وربط حدث <نقر بالزر الأيمن> :
st.text.tag_configure("titre", foreground ="brown",
                      font ="Helvetica 11 bold italic")
st.text.tag_configure("lien", foreground ="blue",
                      font ="Helvetica 11 bold")
st.text.tag_configure("cible", foreground ="forest green",
```

```
41#                        font ="Times 11 bold")
42#  st.text.tag_bind("lien", "<Button-3>", chercheCible)
43#
44#  titre ="""Le Corbeau et le Renard
45#  par Jean de la Fontaine, auteur français
46#  \n"""
47#  auteur ="""
48#  Jean de la Fontaine
49#  écrivain français (1621-1695)
50#  célèbre pour ses Contes en vers,
51#  et surtout ses Fables, publiées
52#  de 1668 à 1694."""
53#
54#  # تعبئة الودجة نص (طريقتين) :
55#  st.importFichier("CorbRenard.txt", encodage ="Latin1")
56#  st.text.insert("0.0", titre, "titre")
57#  st.text.insert(END, auteur, "cible")
58#  # إضافة صورة :
59#  photo =PhotoImage(file= "penguin.gif")
60#  st.text.image_create("6.14", image =photo)
61#  # إضافة علامات إضافية :
62#  st.text.tag_add("lien", "2.4", "2.23")
63#
64#  fen.mainloop()
```

تعليقات

- الأسطر من 3 إلى 6 : الودجة المركب الذي نعرفه في هذا الصنف سيتم اشتقاقه مرة أخرى من صنف **Frame()**. المنشئ ينتظر بعض برامترات المثيل على سبيل المثال (الخط المستخدم، الطور والعرض)، مع قيم الافتراضية. هذه البرامترات ستكون بسيطة التمرير لودجة النص "الداخلي" (الأسطر 7 و 8). ويمكنك بالطبع إضافة العديد من الأشياء الأخرى، لتحديد مظهر لون خلفية المؤشر، ولون الخلفية أو الحروف، وما إذا كان يجب أن تقطع الأسطر الطويلة أو لا ...إلخ. كما يمكنك إرسال برامترات مختلفة إلى شريط التمرير.
- الأسطر من 7 إلى 10 : كما شرحنا سابقا (لودجة مكعب بوكس)، فإنه يأخذ ثلاثة أسطر من التعليمات لإنشاء تبادل بين ويدجات شريط التمرير والنص. بعد تمثيل الودجة النص في السطران 7 و 8، نقوم بصنع شريط تمرير في السطر 9، مع تحديد في التعليمة بتمثيل أسلوب الودجة النص الذي سيصبح تحت سيطرة شريط التمرير. ثم نقوم بإعادة تمثيل ودجة نص في السطر 10، للإشارة إلى أسلوب العودة لشريط التمرير والذي سيتم استدعاؤه للحفاظ على الإرتفاع الصحيح. إعتمادا على شريط تمرير النص الأصلي. وليس من الممكن الإشارة لهذا المرجع عند إنشاء ودجة نص في السطرين 7 و 8، لأن في تلك اللحظة لم يكن شريط التمرير موجودا بعد.
- السطران 11 و 12 : الخيار **expand** للأسلوب **pack()** لا يقبل إلا قيم **YES** أو **NO**. وهي تحدد ما إذا كان يجب أن يمتد الودجة عند تغيير النافذة. خيار المكمل **fill** يمكنه أن يأخذ القيم الثلاثة التالية : **X**، **Y** أو **BOTH**. فهو يشير إلى ما إذا كان الامتداد يتم تنفيذه أفقيا (المحور X) أو عموديا (المحور Y) أو في الاتجاهين (BOTH). عندما تطور تطبيق،

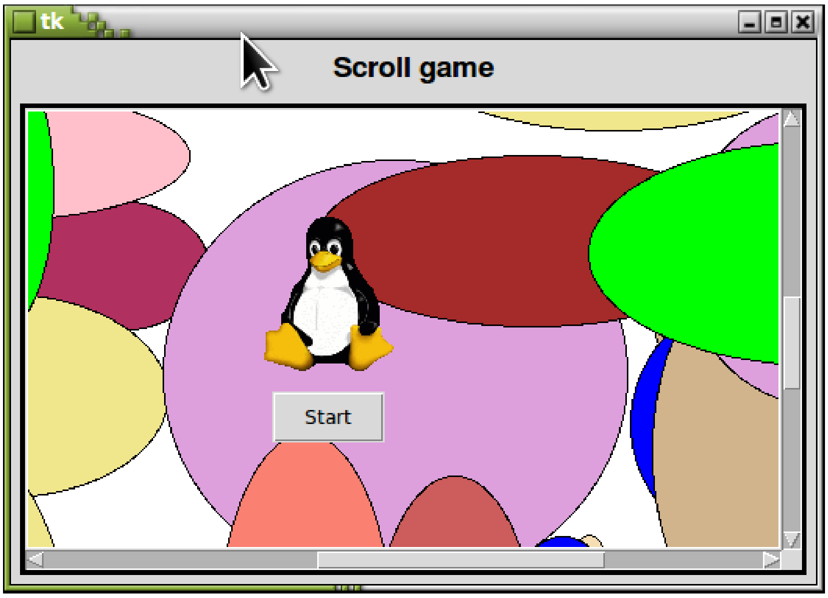
فمن المهم أن تفكر في إعادة تحجيم الصفحة، خاصة إذا كان التطبيق يعمل في أنظمة تشغيل مختلفة (ويندوز، لينكس، ماك).

- الأسطر من 22 إلى 25 : هذه الدالة هي معالج الأحداث، الذي يتم استدعاؤه كلما ضغط المستخدم الزر الأيمن على اسم الكاتب (يتم ربط الحدث مع العلامة المقابلة، في السطر 42).
- في السطر 24، قمنا باستخدام الأسلوب **tag_nextrange()** لودجة النص للعثور على مؤشر لجزء معين من النص المرتبط مع العلامة "الهدف". يقتصر البحث عن هذه المؤشرات على نطاق المعرف في البرامتر الثاني والثالث (في مثالنا، نبحث من بداية إلى نهاية النص). الأسلوب **tag_nextrange()** تقوم بإرجاع مصفوفة مغلقة بها مؤشران (وهي أول وأخر حروف من جزء النص المرتبط بعلامة "الهدف"). في السطر 25، نحن نستخدم واحدة من هذه المؤشرات (الأول) لتفعيل الأسلوب وهذا يؤدي إلى تحريك التلقائي للنص (تمرير)، بحيث الحرف المقابل للمؤشر المرر يكون مرئيا في الودجة (عادة مع عدد من الأحرف التي تتبعها).
- الأسطر من 27 إلى 33 : البناء الكلاسيكي للنافذة يحتوي على ودجةين.
- الأسطر من 35 إلى 42 : هذه الأسطر تعرف ثلاثة علامات **lien**، **titre** و **cible** وهي تربط بتنسيق النص. السطر 42 يحدد أيضا أن النص المرتبط بعلامة **lien** سيكون قابلا للنقر (الزر رقم 3 في الفأرة هو الزر الأيمن)، مع الإشارة لمعالج الأحداث المطابق.
- السطر 55 : يمكنك إدخال أي ميزة في تعريف النص، كما فعلنا هنا من خلال توفير أسلوب لاستيراد الملف النصي في الودجة نفسه، مع البرامتر لفك التشفير. مع هذا الأسلوب، النص الذي تم إستدعائه سيتم إضافته في نهاية النص الذي تم وضعه في الودجة، لكن يمكنك بسهولة أن تحسنها بإضافة برامتر جديد لتحديد الموقع الدقيق حيث يتم إدراجه.
- السطران 56 و 57 : هذه التعليمات تقوم بإدراج أجزاء النص (في بداية ونهاية النص السابق)، بربط علامة مع كل واحدة منهم.
- السطر 62 : ربط العلامات للنص ديناميكي. في أي وقت، يمكنك تفعيل ربط جديد (كما فعلنا هنا من خلال ربك العلامة " lien" لجزء الحالي من النص). لـ"فصل" علامة، استخدم الأسلوب **tag_delete()**.

لوحات مع أشرطة تمرير

لدينا العديد من استغلالات الودجة اللوحة، والإمكانيات واسعة جدا. لقد رأينا كيفية إثراء هذا الصنف بالاشتقاق. وهذا ما سوف نفعله مرة أخرى في المثال أدناه، مع تعريف صنف جديد **ScrolledCanvas**، الذي ربطنا مع لوحة القياسية شريطي تمرير (أفقي وعمودي).

لجعل التمرين أكثر جاذبية، سوف نستخدم صنف ودجة جديد لجعل علبة العنوان، والتي يجب على المستخدم أن ينجح بالضغط على زر الذي يتفاداه باستمرار.



الودجة لوحة متعددة جدا : يسمح بالجمع بين الرسومات والصور النقطية (bmp)، وأجزاء النص والعديد من الويدجات الأخرى، في مساحة ممتدة. إذا كنت ترغب في تطوير لعبة رسومية، فهذه القطعة الأولى التي يجب أن تتقنها.

تطبيقنا الصغير يقدم على أنه صنف جديد **FenPrinc**()، الذي تم الحصول عليه من خلال اشتقاق من صنف الأصل **Tk**(). وهي تحتوي على ودجتين / ملصق بسيط وودجتنا المركب الجديد **ScrolledCanvas**. وهذا هو نظرة للوحة رسم كبيرة جدا، ويمكننا فيها "التجوال" من خلال شريط التمرير.

المساحة المتاحة مرصودة جيدا، بدأ بزرع ديكور بسيط، مصنوع من 80 دائرة بيضاوية من لون والموقع والحجم عشوائي. ولقد قمنا بإضافة غمزة عينة صغيرة في شكل صور نقطية، تهدف أن تذكركم كيفية التعامل مع هذا النوع من الموارد.

و أخيرا، لقد قمنا بتثبيت قطعة وظيفية حقيقية : وهي زر بسيط، لكن تقنية التنفيذ يمكن أن تطبق على أي نوع آخر من الويدجات، بما في ذلك ويدجات مركبة كبيرة مثل التي برمجناها سابقا. هذه المرونة في تطوير التطبيقات المعقدة هي واحدة من الفوائد الرئيسية من البرمجة الشيئية.

الزر يتحرك بأسرع من المرة الأولى، والرسوم المتحركة تتوقف إذا أتينا للضغط على زر جديد. في تحليل النص بالأسفل، أولِي اهتماما بالأساليب المستخدمة لتعديل خصائص كائن موجود .

```
from tkinter import *
from random import randrange

class ScrolledCanvas(Frame):
    """Canevas extensible avec barres de défilement"""
    def __init__(self، boss, width =100, height =100, bg="white", bd=2,
                 scrollregion =(0, 0, 300, 300), relief=SUNKEN):
        Frame.__init__(self, boss, bd =bd, relief=relief)
        self.can =Canvas(self, width=width-20, height=height-20, bg=bg,
                         scrollregion =scrollregion, bd =1)
        self.can.grid(row =0, column =0)
        bdv =Scrollbar(self, orient =VERTICAL, command =self.can.yview, bd =1)
        bdh =Scrollbar(self, orient =HORIZONTAL, command =self.can.xview, bd =1)
        self.can.configure(xscrollcommand =bdh.set, yscrollcommand =bdv.set)
        bdv.grid(row =0, column =1, sticky = NS)       # sticky =>
        bdh.grid(row =1, column =0, sticky = EW)       # رسم الشريط
        # ربط حدث <إعادة التحجيم> إلى المعالج المناسب :
        self.bind("<Configure>", self.redim)
        self.started =False

    def redim(self, event):
        "opérations à effectuer à chaque redimensionnement du widget"
        if not self.started:
            self.started =True       # لا تغير الحجم
            return                   # (عند إنشاء الودجة) و إلا => الحلقة
        # من حجم الإطار الجديد, غير حجم اللوحة
        # (فرق 20 بيكسل للتعويض عن سمك أشرطة التمرير) :
        larg, haut = self.winfo_width()-20, self.winfo_height()-20
        self.can.config(width =larg, height =haut)

class FenPrinc(Tk):
    def __init__(self):
        Tk.__init__(self)
        self.libelle =Label(text ="Scroll game", font="Helvetica 14 bold")
        self.libelle.pack(pady =3)
        terrainJeu =ScrolledCanvas(self, width =500, height =300, relief=SOLID,
                                   scrollregion =(-600,-600,600,600), bd =3)
        terrainJeu.pack(expand =YES, fill =BOTH, padx =6, pady =6)
        self.can =terrainJeu.can
        # ديكور : سلسلة من الأشكال البيضاوية العشوائية :
        coul =('sienna','maroon','brown','pink','tan','wheat','gold','orange',
               'plum','red','khaki','indian red','thistle','firebrick',
               'salmon','coral','yellow','cyan','blue','green')
        for r in range(80):
            x1, y1 = randrange(-600,450), randrange(-600,450)
            x2, y2 = x1 +randrange(40,300), y1 +randrange(40,300)
            couleur = coul[randrange(20)]
            self.can.create_oval(x1, y1, x2, y2, fill=couleur, outline='black')
        # إضافة صورة GIF صغيرة :
        self.img = PhotoImage(file ='linux2.gif')
        self.can.create_image(50, 20, image =self.img)
        # زر للقبض عليه :
        self.x, self.y = 50, 100
        self.bou = Button(self.can, text ="Start", command =self.start)
        self.fb = self.can.create_window(self.x, self.y, window =self.bou)

    def anim(self):
        if self.run ==0:
            return
        self.x += randrange(-60, 61)
        self.y += randrange(-60, 61)
```

```
62#         self.can.coords(self.fb, self.x, self.y)
63#         self.libelle.config(text = 'Cherchez en %s %s' % (self.x, self.y))
64#         self.after(250, self.anim)
65#
66#     def stop(self):
67#         self.run =0
68#         self.bou.configure(text ="Start", command =self.start)
69#
70#     def start(self):
71#         self.bou.configure(text ="Attrapez-moi !", command =self.stop)
72#         self.run =1
73#         self.anim()
74#
75# if __name__ =="__main__":                  # --- برنامج التجربة ---
76#     FenPrinc().mainloop()
```

تعليقات

- الأسطر من 6 إلى 10 : مثل الكثيرة الأخرى، ودجتنا مشتق من **Frame()**. منشئه يقبل عدد من البرامترات. لاحظ أن هذه البرامترات تمرر إلى جزء الإطار (البرامتر bd و relief)، ولجزء اللوحة (البرامترات width و height و bg و scrollregion). يمكنك اختيار اختيارات أخرى، الخيار **scrollregion** لودجة اللوحة يستخدم لتحديد مساحة الرسم في عرض اللوحة التي يمكن أن تتحرك .
- الأسطر من 11 إلى 16 : نحن استخدمنا هذه المرة الأسلوب **grid()** لوضع اللوحة وشريطي التمرير في أماكنهم (هذا الأسلوب قدم لك سابقا في صفحة 101). الأسلوب **pack()** غير مناسب لوضع شريطي التمرير في أماكنها، لأنها تتطلب استخدام العديد من الإطارات المتداخلة (حاول، اعتباره تمرينًا!). التفاعلات بين شريط التمرير والودجة الذي يسيطر عليها (الأسطر 12 و 13 و 14) تم وصفهم بالتفصيل للودجتان المركبان السابقان. الخيار **orient** لشريطي التمرير لم تستخدم حتى الآن، لأن قيمتها الافتراضية (**VERTICAL**) تسبب حالات معالجات.

في الأسطر 15 و 16 الخيارات **sticky =NS** و **sticky =EW** ستسبب امتداد شريطي العنوان إلى فوق كله (**NS** = معناها اتجاه شمال-جنوب) أو العرض (**EW** = معناه الإتجاه شرق-غرب) خلية في الشبكة. سيكون هنالك إعادة تحجيم تلقائية، كما هو الحال مع الأسلوب **pack()** (الخيارات **expand** و **fill** غير متوفرين).

- السطر 18 : منذ أن الأسلوب **grid()** لا يتضمن إعادة التحجيم التلقائية، يجب علينا أن نراقب الحدث الذي يتم إنشاؤه من قبل نظام التشغيل عندما يقوم المستخدم بتغيير حجم الودجة، وربطها مع الأسلوب المناسب لجعلنا نعيد تحجيم مركبات الودجة.
- الأسطر من 19 إلى 29 : أسلوب إعادة التحجيم سيقوم بإعادة تغيير حجم اللوحة (أشرطة التمرير تتكيف بنفسها، بسبب الخيار sticky المطبق). لاحظ أنه يمكنك إيجاد أبعاد الويدجات في سماته **winfo_width()** و **winfo_height()**.

متغير المثيل **self.started** هو مبدل بسيط، يسمح بمنع ما يسمى إعادة التحجيم قبل الأوان، عند تمثيل ودجة (هذا ينتج حلقة غريبة : حاول بدونه) .

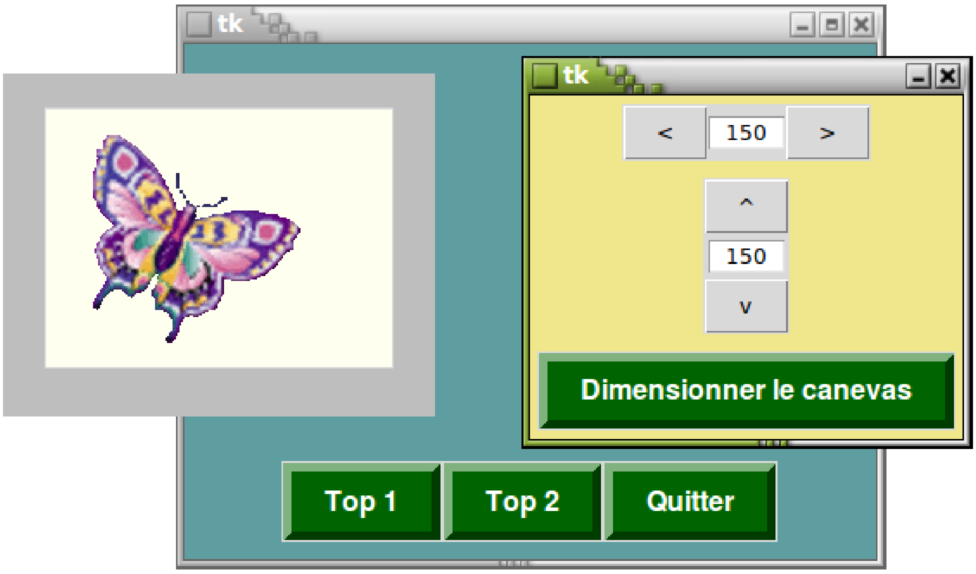
- الأسطر من 31 إلى 55 : هذا الصنف يعرف لعبتنا الصغيرة. منشئها يمثل ودجةنا الجديد في متغير **terrainJeu** (السطر 36). لاحظ أن نوع وسمك الحدود تطبق على إطار الودجة المركب، في حين أن البرامترات الأخرى تختار أنها تطبق على لوحة. مع الخيار scrollregion، عرفنا مساحة اللعبة أكبر بكثير من مساحة سطح اللوحة نفسها، مما يضطر اللاعب للانتقال (أو تغيير حجمها).
- السطران 54 و 55 : الأسلوب **create_window()** لودجة اللوحة هو الذي يسمح بإدخال أي ودجة آخر (بما في ذلك الويدجات المركبة). الودجة الذي يتم إدخاله يجب أن يكون معرف على أنه تابع اللوحة أو النافذة الرئيسية. الأسلوب **create_window()** ينتظر ثلاثة برامترات : الإحداثيات **X** و **Y** للنقطة التي تريد إدراج الودجة ومرجع هذا الودجة .
- الأسطر من 57 إلى 64 : هذا الأسلوب يستخدم لجعل الزر رسم متخرك. بعد تغيير موقع زر تلقائيا عل مسافة معينة من الموقع السابق، وسوف تعيد هذا بعد توقف مؤقت لمدة 250 ميلي ثاني. هذا الإغلاق يحدث بإستمرار، مادام المتغير **self.run** يحتوي على قيمة ليست لا شيء.
- الأسطر من 66 إلى 73 : هذان معالجا الأحداث مرتبطان بزر بالتناوب. فمن الواضح أنه لبدء وإيقاف الرسوم المتحركة .

تطبيق نوافذ متعددة - تعيين ضمني

الصنف **Toplevel()** لـ *tkinter* يسمح بصنع نوافذ "أقمار إصناعية" لتطبيقك الأصل. هذه النوافذ مستقلة، لكنها تغلق تلقائيا عند إغلاق النافذة الرئيسية. ضع هذا القيد على الجانب، ويتم التعامل معها بالطريقة المعتادة : يمكن وضع أي مجموعة من الويدجات.

التطبق الصغير أدناه يوضح لك بعض قدراتها. وهي تتألف من نافذة رئيسية عادية جدا، تحتوي على 3 أزرار. هذه الأزرار صنعت بصنف مشتق من صنف **Button()** أساسا، وذلك ليظهر لك مرة أخرى كم هو من السهل تكيف أصناف الكائن الموجود لأجلك. سوف تلاحظ بعض خيارات "الديكور" المثيرة للاهتمام.

الزر <Top1> يظهر لك أول نافذة قمر صناعي تحتوي على لوحة مع سورة. لدينا مع هذه النافذة خصائصها الخاصة : وهي ليس لديها لا شعار-عنوان ولا حدود، ومستحيل إعادة تحجيمها بواسطة الفأرة. بالإضافة إلى ذلك، هذه النافذة مشروطة : الأمر يستحق كل نافذة لا تزال في المقدمة، أمام جميع النوافذ الأخرى لتطبيقات أخرى قد تكون موجودة على الشاشة .



الزر <Top2> يعرض نافذة قمر صناعي أكثر كلاسيكية، تحتوي على نسختين من ودجة مركب صغير **SpinBox** صنعناه وفقا للمبادئ المذكورة في الصفحات السابقة. هذا الودجة يتكون من زران وملصق يشير إلى قيمة رقمية. الأزرار تسمح بزيادة أو تقليل القيمة المعروضة. بالإضافة إلى هذان SpinBoxes، النافذة تحتوي على زر كبير مزين. عند تشغيله، المستخدم يقوم بتغيير حجم اللوحة في نافذة قمر صناعي أخرى، بالاتفاق مع القيم الرقمية المعروضة في 2 SpinBoxes .

```
from tkinter import *

class FunnyButton(Button):
    "Bouton de fantaisie : vert virant au rouge quand on l'actionne"
    def __init__(self, boss, **Arguments):
        Button.__init__(self, boss,  bg ="dark green", fg ="white", bd =5,
                        activebackground ="red", activeforeground ="yellow",
                        font =('Helvetica', 12, 'bold'), **Arguments)

class SpinBox(Frame):
    "widget composite comportant un Label et 2 boutons 'up' & 'down'"
    def __init__(self, boss, largC=5, largB =2, vList=[0], liInd=0, orient =Y):
        Frame.__init__(self, boss)
        self.vList =vList           # قائمة القيم المتوفرة
        self.liInd =liInd           # مؤشر قيمة العرض الإفتراضي
        if orient ==Y:
            s, augm, dimi = TOP, "^", "v"       # التوجيه 'عمودي'
        else:
            s, augm, dimi = RIGHT, ">", "<"     # التوجيه 'أفقي'
        Button(self, text =augm, width =largB, command =self.up).pack(side =s)
        self.champ = Label(self, bg ='white', width =largC,
                           text =str(vList[liInd]), relief =SUNKEN)
        self.champ.pack(pady =3, side =s)
        Button(self, text=dimi, width=largB, command =self.down).pack(side =s)

    def up(self):
        if self.liInd < len(self.vList) -1:
```

```
            self.liInd += 1
        else:
            self.bell()       # صوت تنبيه
        self.champ.configure(text =str(self.vList[self.liInd]))

    def down(self):
        if self.liInd > 0:
            self.liInd -= 1
        else:
            self.bell()       # صوت تنبيه
        self.champ.configure(text =str(self.vList[self.liInd]))

    def get(self):
        return self.vList[self.liInd]

class FenDessin(Toplevel):
    "Fenêtre satellite (modale) contenant un simple canevas"
    def __init__(self, **Arguments):
        Toplevel.__init__(self, **Arguments)
        self.geometry("250x200+100+240")
        self.overrideredirect(1)            # => نافذة دون حدود
        self.transient(self.master)         # => نافذة 'modale'
        self.can =Canvas(self, bg="ivory", width =200, height =150)
        self.img = PhotoImage(file ="papillon2.gif")
        self.can.create_image(90, 80, image =self.img)
        self.can.pack(padx =20, pady =20)

class FenControle(Toplevel):
    "Fenêtre satellite contenant des contrôles de redimensionnement"
    def __init__(self, boss, **Arguments):
        Toplevel.__init__(self, boss, **Arguments)
        self.geometry("250x200+400+230")
        self.resizable(width =0, height =0)    # => منع تغيير
        p =(10, 30, 60, 90, 120, 150, 180, 210, 240, 270, 300)
        self.spX =SpinBox(self, largC=5,largB =1,vList =p,liInd=5,orient =X)
        self.spX.pack(pady =5)
        self.spY =SpinBox(self, largC=5,largB =1,vList =p,liInd=5,orient =Y)
        self.spY.pack(pady =5)
        FunnyButton(self, text ="Dimensionner le canevas",
                    command =boss.redimF1).pack(pady =5)

class Demo(Frame):
    "Démo. de quelques caractéristiques du widget Toplevel"
    def __init__(self):
        Frame.__init__(self)
        self.master.geometry("400x300+200+200")
        self.master.config(bg ="cadet blue")
        FunnyButton(self, text ="Top 1", command =self.top1).pack(side =LEFT)
        FunnyButton(self, text ="Top 2", command =self.top2).pack(side =LEFT)
        FunnyButton(self, text ="Quitter", command =self.quit).pack()
        self.pack(side =BOTTOM, padx =10, pady =10)

    def top1(self):
        self.fen1 =FenDessin(bg ="grey")

    def top2(self):
        self.fen2 =FenControle(self, bg ="khaki")

    def redimF1(self):
        dimX, dimY = self.fen2.spX.get(), self.fen2.spY.get()
        self.fen1.can.config(width =dimX, height =dimY)
```

```
89#
90# if __name__ =="__main__":                # --- برنامج التجربة ---
91#     Demo().mainloop()
```

تعليقات

• الأسطر من 3 إلى 8 : إذا كنت ترغب في الحصول على نفس تصميم الأزرار المختلفة في الأجزاء المختلفة من مشروعك، لا تتردد في صنع فئة مشتقة، كما فعلنا هنا. وهذا سيوفر لك الحاجة لإعادة برمجة نفس الخيارات المحددة.

لاحظ النجمتين التي بدأنا بها اسم نهاية برامتر المنشئ: **Arguments** (**برامترات). وهذا يعني أن هذا المتغير هو قاموس، قادر على تلقي تلقائيا أي مجموعة من البرامترات مع الملصق. هذه البرامترات يمكنها تمرير نفسها للمنشئ الصنف الأصل (في السطر 8). وهذا يعفينا من الاضطرار إلى إعادة كتابة جميع الخيارات البرامترية للزر الأساس في قائمة البرامترات ، والتي هي كثيرة جدا. وأيضا يمكنك تمثيل هذه الأزرار مع أي مجموعة من الخيارات، طالما أنه متاحة للأزرار الأساس. ونسمي هذا بالبرامترات الضمنية. يمكنك استخدام هذا الشكل من البرامترات مع أي دالة أو أسلوب.

• * الأسطر من 10 إلى 24 : منشئ الودجة الخاص بنا **SpinBox** لا يحتاج إلا القليل من التعليقات. إعتمادا على التوجيه المطلوب، الأسلوب **pack()** يتيح أزرار وملصقات من الأعلى للأسفل أو اليمين لليسار (البرامترات **TOP** أو **RIGHT** للخيار **side**).

• الأسطر من 26 إلى 38 : هذان الأسلوبان لا يفعلان شيئا أكثر من تعديل قيمة المعروضة في الملصق. لاحظ أن الصنف **Frame()** لديه أسلوب **bell()** ليتسبب في صوت "بيب".

• * الأسطر من 43 إلى 53 : تعريف نافذة القمر الصناعي الأولى هنا. لاحظ مرة أخرى استخدام البرامتر الضمني للمنشئ بمساعدة المتغير **Arguments**. هذا هو الذي يسمح لنا بتمثيل هذه النافذة (في السطر 81) عن طريق تحديد لون الخلفية (يمكن أن يطلب أيضا الحدود، إلخ ...). أساليب الاستدعاء في الأسطر من 47 إلى 49 تم تعريف بعض خصائصه (ينطبق على أي نافذة). الأسلوب **geometry()** يسمح لك بتعيين إحداثيات النافذة ومكانه في النافذة

(100+240+) تعني أنه ينبغي تحويل الزاوية العلوية اليسرى 100 بيكسل إلى اليمين و 240 بيكسل أسفل زاوية اليسرى العلوية من الشاشة) .

• الأسطر 45 و 57 : يرجى ملاحظة الفرق الصغير بين قوائم البرامترات لهذه الأسطر. في منشئ **FenDessin** تم حذف البرامتر **boss** الموجود في منشئ **FenControle**. هل تعلم أن هذا البرامتر يستخدم لتمرير مرجع الودجة "السيد" لـ "عبيده". وهو عموما ضروري (في السطر 67، على سبيل المثال، فإننا استخدمنا للمرجع أسلوب البرنامج الرئيسي)، لكن لم يكن ضروريا للغاية: في **FenDessin** ليس لدينا به أي فائدة. سوف تجد الفرق واضح في تعليمات التمثيل لهذه النوافذ، في الأسطر 82 و 84.

• الأسطر من 55 إلى 67 : باستثناء الفرق المذكور أعلاه، منشئ الودجة **FenControle** مشابهة جدا لـ **FenDessin**. لاحظ في السطر 60 الأسلوب الذي يسمح بمنع تغيير حجم النافذة (في حالة النافذة بدون حدود وبدون إعلان-العنوان، مثل **FenDessin**، سيكون هذا الأسلوب بدون جدوى).

• الأسطر 73 و 74 (و 49) : جميع الأصناف المشتقة هي ويدجات tkinter مميزة تلقائيا بسمة **master**، الذي يحتوي على مرجع الصنف الأصل. الذي يسمح لنا بالوصول إلى أبعاد ولون خلفية من النافذة الرئيسية.

• *الأسطر من 86 إلى 88 : هذا الأسلوب يسترد القيم الرقمية المعروضة في نافذة التحكم (الأسلوب **get**() لهذا الودجة)، لإعادة تحجيم لوحة نافذة الرسم. هذا المثال البسيط يبين لك، مرة أخرى، كيفية تحقيق تواصل بين مختلف مكونات البرنامج .

أشرطة الأدوات - تعبير لامدا (lambda)

العديد من البرامج لديها شريط أدوات (تولبار - toolbar) أو أكثر تتكون من أزرار صغيرة تمثل أيقونات. هذا النهج يتيح لك أن تقدم للمستخدم عددا كبيرا من الأوامر الخاصة، وليس لأنها تحتل مساحة على الشاشة أيضا .



البرنامج الموضح أدناه لديه شريط أدوات ولوحة. عندما يضغط المستخدم على واحدة من 8 أزرار في الشريط، يتم نسخ الأيقونة على اللوحة، على موقع مختار عشوائيا. عندما يتم النقر على الزر الأخير، يتم حذف محتوى اللوحة .

```
from tkinter import *
from random import randrange

class ToolBar(Frame):
    "Barre d'outils (petits boutons avec icônes)"
    def __init__(self, boss, images =[], command =None, **Arguments):
        Frame.__init__(self, boss, bd =1, **Arguments)
        # <images> = قائمة أسماء الأيقونات لوضعها على الأزرار
        self.command =command          # أمر لتنفيذه عند النقر
        nBou =len(images)              # عدد الأزرار لبنائها
        # يجب وضع الأيقونات في متغيرات ثابة
        # قائمة تقوم بهذا :
        self.photoI =[None]*nBou
        for b in range(nBou):
            # إنشاء أيقونة (objet PhotoImage Tkinter) :
```

```
            self.photoI[b] =PhotoImage(file = images[b] +'.gif')
            # إنشاء زر. نقوم باستدعاء دالة lambda
            # لتمرير برامتر إلى الأسلوب <action> :
            bou = Button(self, image =self.photoI[b], bd =2, relief =GROOVE,
                         command = lambda arg =b: self.action(arg))
            bou.pack(side =LEFT)

    def action(self, index):
        # تنفيذ <command> مع مؤشر الزر كبرامتر :
        self.command(index)

class Application(Frame):
    def __init__(self):
        Frame.__init__(self)
        # أسماء الملفات التي تحتوي على الأيقونات (نوع GIF):
        icones =('floppy_2', 'coleo', 'papi2', 'pion_1', 'pion_2', 'pion_3',
                 'pion_4', 'help_4', 'clear')
        # إنشاء شريط أدوات :
        self.barOut =ToolBar(self, images =icones, command =self.transfert)
        self.barOut.pack(expand =YES, fill =X)
        # إنشاء لوحة لاستقبال الصور :
        self.ca = Canvas(self, width =400, height =200, bg ='orange')
        self.ca.pack()
        self.pack()

    def transfert(self, b):
        if b ==8:
            self.ca.delete(ALL)       # مسح كل شيء في اللوحة
        else:
            # قم بنسخ أيقونة الزر b (المستخرج من الشريط) => اللوحة :
            x, y = randrange(25,375), randrange(25,175)
            self.ca.create_image(x, y, image =self.barOut.photoI[b])

Application().mainloop()
```

ميتابروغراميك - التعبير لامدا

أنت تعلم بصفة عامة، يرتبط كل زر بأمر ما، وهو مرجع لأسلوب أو دالة معينة تقوم بالعمل عند تنشيط الزر. ومع ذلك، في هذا تطبيق يعرض، جميع الأزرار التي تقوم بنفس الشيء تقريبا (نسخ الرسم على لوحة)، الفرق الوحيد بينها هو رسم concerné.

لتبسيط كودنا، نحن نريد ربط الخيار **command** لكل الأزرار مع واحد ونفس الأسلوب (سيكون الأسلوب **action()**)، لكن في كل مرة نقوم فيها بتمرير المرجع للزر المستخدم بشكل خاص، بحيث يكون نشاطه مختلفا عن كل واحد فيهم.

سوف تنشأ صعوبة، لأن الخيار **command** لودجة الزر يقبل فقط قيمة أو تعبير، وليس تعليمة. بل هو في الواقع يشير إلى مرجع الدالة، لكن ليس لاستدعاء دالة مع هذه البرامترات (الاستدعاء سيتم في الواقع عندما يقوم المستخدم بالضغط على الزر : وهذا هو متلقي أحداث لـ tkinter). هذا هو سبب وجود اسم الدالةفقط ، بدون أقواس.

يمكننا حل هذه المشكلة بطريقتين :

- و نظرا لطبيعته الديناميكية، يقبل بيثون البرنامج الذي يمكنه تغيير نفسه، على سبيل المثال من خلال تعريف وظائف جديد أثناء التنفيذ (و هذا هو مفهوم الميتابروغراميك).

و لذلك من الممكن تعرف دالة مع برامترات، يشير لك واحد منها قيمة افتراضية، ثم توفير مرجع لهذه الدالة للخيار **command**. منذ أن يتم تعريف دالة أثناء التشغيل، هذه القيم الافتراضية يمكن أن تكون ضمن محتويات المتغير. عندما يمكن للحدث "الضغط على زر" استدعاء دالة، لذلك سوف تستخدم القيم الافتراضية لبرامتراته، كما لو كانت برامترات. الناتج من العملية هو بالتالي تمرير برامترات كلاسيكية.

- لتوضيح هذه التقنية إستبدل السطر من 17 إلى 20 من السكريبت بهذا :

```
# إنشاء زر  بتعريف دالة لحظيا مع برامتر ذا قيمة إفترضية
: و هي البرامتر المرر. هذه الدالة تقوم باستدعاء الأسلوب الذي يتطلب برامتر
def agir(arg = b):
    self.action(arg)
# الأمر يرتبط مع زر يستدعي الدالة أدناه :
bou = Button(self, image = self.photoI[b], relief = GROOVE,
             command = agir)
```

- كل الذي سبق يمكن تبسيطه، باستخدام تعبير لامدا. هذه الكلمة المحجوزة في بيثون هي تعبير يقوم بإرجاع كائن دالة، مماثلة لتلك التي تقوم بإنشاء مع التعليمة **def**. لكن مع إختلاف أن لامدا هي تعبير وليس تعليمة، يمكننا استخدامه كواجهة لاستدعاء دالة (مع تمرير برامترات)وهو عادة غير ممكن. لاحظ أن مثل هذه الدالة مجهولة (أي ليس لها اسم).

على سبيل المثال، التعبير :

```
lambda ar1=b, ar2=c : bidule(ar1,ar2)
```

يقوم بإرجاع مرجع دالة مجهولة تم صنعها، والتي تستطيع استدعاء الدالة **bidule()** وتمرير برامترا **b** و **c**، وتستخدم الأخيرة قيما افتراضية في تعريف البرامترات **ar1** و **ar2** للدالة.

هذه التقنية تستخدم نفس المبدأ السابق، لكن لديها ميزة كونها أكثر إيجازا، وهو سبب في إننا قد استخدمناها في سكريبتنا. ومع ذلك، فإنه من الصعب فهمها :

```
command = lambda arg =b: self.action(arg)
```

في هذا الجزء من التعليمات، الأمر يرتبط مع الزر الذي هو مرجع للدالة المجهولة مع تمرير للبرامتر **arg** قيمة أولية : قيمة البرامتر هي **b**.

يتم استدعاؤها بدون برامتر من الأمر، هذه الدالة المجهولة يمكنها القيام بنفس استخدام البرامتر **arg** (مع قيمة افتراضية) باستدعاء للأسلوب الهدف **self.action()**، ونحصل على نقل برامتر حقيقي لهذا الأسلوب.

نحن هنا لا نريد تفاصيل السؤال عن التعبير لامدا، لأنه خارج الإطار الذي وضعناه للتهيئة. إذا أردت المزيد، يمكنك سؤال أحدهم أو البحث في مراجع اللغة .

تمرير دالة (أو أسلوب) كبرامتر

لقد تحدثنا بالفعل على العديد من الويدجات المركبة والكثير من خياراتها مثل الخيار **command**، الذي يجب أن يرتبط مع اسم الدالة أو الأسلوب. بعبارات أعم، هذا معناه أن الدالة مع البرامتر يمكنها تلقي مرجع دالة أخرى كبرامتر، وفائدة هذا الشيء واضحة هنا.

و لقد قمنا ببرمجة وظيفة مميزة مثل هذه في صنفنا الجديد **ToolBar()**. ويمكنك أن ترى أننا قمنا بإدراج اسم **command** في قائمة البرامترات لمنشئه، في السطر 6. هذا البرامتر ينتظر مرجع دالة أو أسلوب كبرامتر. ثم يتم تخزين المرجع في متغير مثيل (في السطر 9)، بحيث يكون في متناول بقية أساليب النصف. ويمكن لهذا أن تستدعي دالة أو أسلوب (إذا لزم الأمر عن طريق تمرير برامترات، بعد شرح التقنية في الجزء السابق). وهذا ما يفعله أسلوبنا **action()**ـ في السطر 25. في هذه الحال، أسلوب التمرير هو أسلوب **transfert()** للصنف **Application** (انظر للسطر 34).

النتيجة لهذا هو أننا قادرون على تطوير صنف كائنات **ToolBar** قابلة لإعادة الاستخدام في سياقات أخرى. كما ترى تطبيقنا الصغير، يكفي إنشاء مثيل لهذه الكائنات التي تشير إلى مرجع أي دالة ببرامتر الخيار `command`. هذه الدالة سيتم استدعاؤها تلقائيا مع رقم ترتيبها للزر عندما يضغطه المستخدم.

لا تتردد في تخيل ماذا تفعله هذه الدالة !

للانتهاء من هذا المثال، لاحظ تفاصيل أخرى أيضا : يحيط بكل واحد من أزرارنا بأخدود (الخيار `relief =GROOVE`). يمكنك بسهولة الحصول على مناطق أخرى عن طريق خيار **relief** و **bd** (الحافة في تعليمة المثيل لهذه الأزرار). وخاصة، يمكن اختيار `relief =FLAT` و `bd =0` للحصول على أزرار صغيرة "مسطحة"، بدون أي relief .

نوافذ مع قوائم

و لإنهاء زيارتنا الصغيرة لويدجات tkinter، سنقوم الآن بشرح منشئ نافذة للتطبيق مع أنواع مختلفة من القوائم "القاعدة"، كل واحد من هذه القوائم قادر على "الفصل" من التطبيق الرئيسي لتصبح نافذة مستقلة، كما هو مبين على الجانب.



كما شرحنا سابقا[76]، هذا الأسلوب لبدأ كتابة برنامج بمسودة، الذي يحتوي فقط على بضعة أسطر لكنها فعالة. سوف نجرب هذه المسودة بعناية للقضاء على أي علل. عندما تعمل المسودة بشكل جيد، سوف نضيف الوظائف الإضافية. نحن نختبر هذا الملحق حتى يعطي ارتياحا، ثم نضيف جزءا آخر، وهكذا ...

هذا لا يعني أنه لا يمكنك البدء فورا بالبرنامج دون إجراء تحليل دقيق للمشروع، على الأقل يجب أن يكون المخطط موصوفا على نحو كافٍ وبوضوح في كراس المواصفات.

> و من الضروري أن تقوم بوضع تعليق بشكل صحيح على كود المنتج عند تطويره. ونحن نسعى لكتابة تعليقات جيدة وهي في الواقع ضرورية. ليس فقط ليكون كودك أسهل للقراءة (و بالتالي للحفاظ على وقت الآخرين ولك). ولكن أيضا إذا إضطررت للتعبير عن ما تريد حقا أن يفعله (انظر إلى الأخطاء الدلائلية صفحة 7).

مواصفات التمرين

تطبيقنا يحتوي ببساطة على شريط قوائم ولوحة. عناوين مختلفة وخيارات القوائم تؤدي إلى إظهار أجزاء نص في اللوحة أو في تعديل تفاصيل الديكور، ولكن سيكون في البداية مجموعة من الأمثلة، وتهدف إلى إعطائك العديد من الاحتمالات الذي يقدمها هذا النوع من الودجة، الإكسسوارات الأساسية لأي تطبيق حديث الذي لديه بعض الأهمية.

نريد أيضا تنظيف كود البرنامج جيدا في هذا التمرين. للقيام بهذا، سوف نستخدم صنفين : صنف للتطبيق الرئيسي، وأخر لشريط العنوان. ونحن نريد أن نسلط الضوء على منشئ تطبيق يتضمن عدة أصناف كائن تفاعلي .

أول مسودة للبرنامج

عند بناء مسودة لبرنامج، يتعين علينا أن نحاول تجميع الهيكل، مع العلاقات بين الكتل الرئيسية التي تشكل التطبيق النهائي. هذا ما نحاول القيام به في المثال أدناه .

```
1# from tkinter import *
```

[76] انظر الصفحة 8 : البحث عن الأخطاء و التجريب.

```

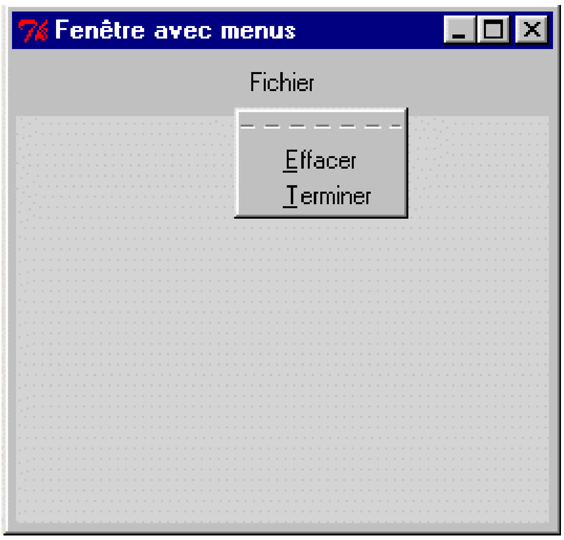
class MenuBar(Frame):
    """Barre de menus déroulants"""
    def __init__(self, boss =None):
        Frame.__init__(self, borderwidth =2)

        ##### <قائمة <ملف #####
        fileMenu = Menubutton(self, text ='Fichier')
        fileMenu.pack(side =LEFT)
        # Partie "déroulante" :
        me1 = Menu(fileMenu)
        me1.add_command(label ='Effacer', underline =0,
                        command = boss.effacer)
        me1.add_command(label ='Terminer', underline =0,
                        command = boss.quit)
        # تكامل القائمة :
        fileMenu.configure(menu = me1)

class Application(Frame):
    """البرنامج الرئيسي"""
    def __init__(self, boss =None):
        Frame.__init__(self)
        self.master.title('Fenêtre avec menus')
        mBar = MenuBar(self)
        mBar.pack()
        self.can = Canvas(self, bg='light grey', height=190,
                          width=250, borderwidth =2)
        self.can.pack()
        self.pack()

    def effacer(self):
        self.can.delete(ALL)

if __name__ == '__main__':
    app = Application()
    app.mainloop()
```



لذا يرجى الآن ترميز هذه الأسطر وتجربة تشغيلها. يجب عليك أن تحصل على نافذة مع لوحة رمادية فاتحة يعلوها شريط القوائم. في هذه المرحلة، شريط القوائم لا يحتوي سوى على قائمة وحيدة "الملف - Fichier".

أنقر على "الملف" لإظهار القائمة : الخيار "مسح - Effacer" لا يعمل بعد (سوف يمحو محتويات اللوحة)، لكن الخيار "إنهاء - Terminer" يجب أن يغلق بالفعل التطبيق بشكل صحيح.

مثل جميع القوائم التي صنعناها بواسطة tkinter، القائمة التي صنعتها يمكنها تحويل إلى قائمة "تعويم" : يكفي أن تضغط ببساطة على سطر المنقط على رأس القائمة. سوف تحصل على نافذة "قمر صناعي" صغيرة، يمكنك وضعها في أي مكان تريد على سطح المكتب .

تحليل السكريبت

هيكل هذا البرنامج الصغير يجب أن يظهر لك المعتاد : إن الفئات التي تم تعريفها في هذا السكريبت يمكن أن تستخدم مرة أخرى في مشاريع أخرى عن طريق الاستيراد، كما شرحناه سابقا[77]، جسم البرنامج الرئيسي (الأسطر من 35 إلى 37) تحتوي على البيان الكلاسيكي: **'__if __name__ == '__main**:

التعليمتان التاليتان تتكون فقط بتمثيل كائن **app** وتشغيل أسلوبه **()mainloop**. كما تعلمون، فإننا يمكننا اختصار هاتين التعليمتين إلى واحدة.

أساس البرنامج تجدها في تعريفات الصنف السابقة.

الصنف **()MenuBar** تحتوي على وصف لشريط القوائم. في هذه الحالة للسكريبت، تتلخص في مسودة المنشئ .

- السطر 5 : البرامتر **boss** يتلقى مرجع النافذة الرئيسية للودجة في لحظة تمثيله. هذا المرجع يسمح لنا باستدعاء الأساليب المرتبطة بهذه النافذة الأساسية، في الأسطر 14و 16.
- السطر 6 : التفعيل الإلزامي لمنشئ الصنف الأصل.
- السطر 9 : تمثيل ودجة صنف **()Menubutton**، سيتم تعرفه على أنه "عبيد" لـ **self** (هذا معناه كائن مركب "شريط القوائم" لذا نحن منشغلون بتحديد الصنف). كما يوحي اسمه، هذا النوع من الودجة يتصرف قليلا مثل الزر : الحركة تعمل عند الضغط على الزر.
- السطر 12 : هذا العمل يعمل على إظهار قائمة حقيقية، يبقى تعريفه : أنه ودجة جديد، للصنف **()Menu** هذه المرة. يتم تعريفه على أنه "عبيد" للودجة **Menubutton** الممثلة في السطر 9.

77انظر الصفحة 202 ، وحدات تحتوي مكتبات الأصناف.

• الأسطر من 13 إلى 16 : يمكننا تطبيق عدد من الأساليب المحددة على الويدجات للصنف **Menu()** كل واحدة يقبل العديد من الخيارات. نحن نستخدم هنا الأسلوب **add_command()** لتثبيت عنصرين في القائمة هما: "مسح - Effacer" و"إنهاء - Terminer". سوف ندمج الآن، الخيار **underline**، الذي يستخدم لتعريف اختصار لوحة المفاتيح : هذا الخيار يشير إلى أي حروف للعنصر يجب أن تظهر مسطرة على الشاشة. المستخدم يعرف أنه إذا ضغط على هذا الحرف من لوحة المفاتيح ليتم تفعيل وظيفة هذا العنصر (كما لو نقرنا بالفأرة).

سيعمل عندما يقوم المستخدم بتحديد عنصر من الخيار **command**. في سكريبتنا الأوامر المعطاة هي أسلوبان للنافذة الأصل، وسوف يتم تمرير الودجة في لحظة تمثيله عن طريق البرامتر **boss**. الأسلوب **effacer()**، الذي سنعرفه لاحقا، يعمل على مسح اللوحة. الأسلوب المعرف مسبقا **quit()** يعمل على الخروج من حلقة **mainloop()** وإيقاف إستقبال الأحداث المرتبطة مع نافذة التطبيق.

• السطر 18 : عندما يتم تعريف عنصر من القائمة، لا يزال بحاجة إلى إعادة تكوين ودجة الأصل **Menubutton** بحيث الخيار "menu - قائمة" هي في الواقع القائمة التي نريد بناها في الواقع نحن لا نستطيع تحديد هذا الخيار عند التعريف الأصلي للودجة **Menubutton**، لأن في هذه المرحلة، القائمة لم تكن موجودة بعد. لا نستطيع أن لا نعرف الودجة **Menu** في المكان الأول، لأنه سيتم تعريفه على أنه "عبيد" لودجة **Menu** يجب أن نفعل ذلك عن طريق 3 خطوات كما فعلنا سابقا، يجب أن نستدعي الأسلوب **Menubutton**. يجب أن نفعل ذلك عن طريق 3 خطوات كما فعلنا سابقا، يجب أن نستدعي الأسلوب **configure()**. هذا الأسلوب يمكن تطبيقه على أي ودجة موجودة لتعديل خيار أو أكثر .

الصنف **Application()** يحتوي على وصف للنافذة الرئيسية وأساليب معالجة الأحداث التي مرتبطة بها .

• السطر 20 : نحن نفضل إشتقاق تطبيقنا من الصنف **Frame()**، الذي لديه خيارات كثيرة، بدلا من الطبقة الأولية **Tk()**. بهذه الطريقة، التطبيق يتم تغليفه في ودجة، والذي قد يكون متكاملا في وقت لاحق في تطبيق أهم. تذكر أن، في أي حال، tkinter يمثل تلقائيا نافذة الأصل من نوع **Tk()** لاحتواء هذا الإطار.

• السطران 23 و 24 : بعد تفعيل اللازمة من المنشئ لصنف الأصل، نحن استخدمنا سمات **master** التي tkinter مرتبطة بها تلقائيا في كل ودجة، لمرجع الصنف لأصل (في الحالة السابقة، الكائن هو النافذة الرئيسية للتطبيق) وإعادة تعريف شعار-العنوان.

• الأسطر من 25 إلى 29 : تمثيل لويجدتان "عبيد" لإطارنا (**Frame**) الرئيسي. من الواضح أن شريط القوائم هو ودجة معرف في صنف آخر.

• السطر 30 : مثل أي قطعة من أي ودجة آخر، إطارنا الرئيسي يجب أن يكون معهودا لأسلوب لتنفيذ إظهار الحقيقية.

• الأسطر 32 و 33 : الأسلوب المستخدم لمسح اللوحة يتم تعرفه في النصف (لأن الكائن لوحة في الحقيقة جزء)، لكن يتم استدعاؤها من خلال الخيار **command** للودجة العبيد الذي تم تعريفه في صنف آخر.

كما شرحنا أعلاه، هذا الودجة يتلقى مرجع لودجة السيد عن طريق البرامتر **boss**. كل هذه المراجع تحدد أولويات بمساعدة إشارات أسماء بالنقاط .

إضافة القائمة Musiciens (الموسيقيين)

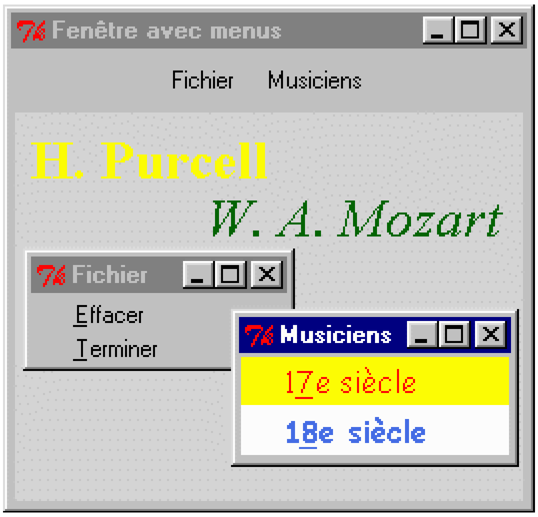
واصل تطوير هذا البرنامج الصغير، بإضافة الأسطر التالية في منشئ الصنف **MenuBar()** (بعد السطر 18):

```
##### <قائمة الموسيقيين> #####
self.musi = Menubutton(self, text ='Musiciens')
self.musi.pack(side =LEFT, padx ='3')
# جزء "أسفل" قائمة الموسيقيين :
me1 = Menu(self.musi)
me1.add_command(label ='17e siècle', underline =1,
                foreground ='red', background ='yellow',
                font =('Comic Sans MS', 11),
                command = boss.showMusi17)
me1.add_command(label ='18e siècle', underline =1,
                foreground='royal blue', background ='white',
                font =('Comic Sans MS', 11, 'bold'),
                command = boss.showMusi18)
# تكامل القائمة :
self.musi.configure(menu = me1)
```

... وتعاريف الأساليب التالية للصنف **Application()** (بعد السطر 33):

```
def showMusi17(self):
    self.can.create_text(10, 10, anchor =NW, text ='H. Purcell',
                font=('Times', 20, 'bold'), fill ='yellow')

def showMusi18(self):
    self.can.create_text(245, 40, anchor =NE, text ="W. A. Mozart",
                font =('Times', 20, 'italic'), fill ='dark green')
```



عندما تقوم بإضافة كل هذه الأسطر، قم بحفظ السكريبت وقم بتشغيله.

شريط القوائم يضم الآن قائمة جديدة : قائمة "Musiciens" (الموسيقيين).

القائمة المقابلة تقدم عنصرين يظهران مع ألوان وخطوط شخصية. يمكنك أن تتعلم هذه التقنيات الزخرفية لمشاريعك الشخصية.

الأوامر التي قمت بربطها مع هذه العناصر هي مبسطة حتى لا نتعب في التمرين : لأنها تسبب عرض نصوص صغيرة على اللوحة .

تحليل السكريبت

التغيرات الوحيدة التي أدخلت في هذه الأسطر هي استخدام خطوط للحروف المحددة (الخيار **font**)، ولون المقدمة (الخيار **foreground**) ولون الخلفية (الخيار **background**) للنصوص المعروضة.

يرجى ملاحظة مرة أخرى أن استخدام الخيار **underline** لتعيين حرف ليكون اختصار للوحة المفاتيح (لا ننسى أن ترقيم الأحرف يبدأ من الصفر)، وخاصة الخيار **command** لهذه الويدجات تصل إلى أساليب للصنف الآخر، من خلال مرجع مخزن في سمة **boss**.

الأسلوب **create_text()** للوحة يجب استخدامه مع برامترين رقميين، واللذان هما الإحداثيات X و Y للنقطة في اللوحة. النص المرر سيتم وضعه في هذه النقطة، في دالة للقيمة المختارة للخيار **anchor** : هنا تم تحديد كيفية كون جزء النص "راسية" في النقطة المختارة في اللوحة، في وسطه أو في الجانب أقصى اليسار أو إلخ ...، في دالة بناء جملة التي تستخدم قياسا على نقاط الأساسية الجغرافية (**NW** = لأقصى اليسار، **SE** = لليسار العلوي، **CENTER** = للوسط، إلخ.).

إضافة قائمة Peintres (الرسامين)

Cette هذه القائمة الجديدة تم صنعها بطريقة مماثلة تماما لسابقتها، لكن نحن أضفنا وظيفة إضافية : القوائم "الشلالات".

يرجى إذا إضافة الأسطر التالية في منشئ الصنف **MenuBar**() :

```
##### <قائمة الرسامين> #####
self.pein = Menubutton(self, text ='Peintres')
self.pein.pack(side =LEFT, padx='3')
# جزء الأسفل :
me1 = Menu(self.pein)
me1.add_command(label ='classiques', state=DISABLED)
me1.add_command(label ='romantiques', underline =0,
                command = boss.showRomanti)
# قائمة فرعية للرسامين الإنطباعيين :
me2 = Menu(me1)
me2.add_command(label ='Claude Monet', underline =7,
                command = boss.tabMonet)
me2.add_command(label ='Auguste Renoir', underline =8,
                command = boss.tabRenoir)
me2.add_command(label ='Edgar Degas', underline =6,
                command = boss.tabDegas)
# تكامل القائمة الفرعية :
me1.add_cascade(label ='impressionistes', underline=0, menu =me2)
# تكامل القائمة :
self.pein.configure(menu =me1)
```

... وتعاريف الأساليب التالية للصنف **Application**() :

```
def showRomanti(self):
    self.can.create_text(245, 70, anchor =NE, text = "E. Delacroix",
                font =('Times', 20, 'bold italic'), fill ='blue')

def tabMonet(self):
    self.can.create_text(10, 100, anchor =NW, text = 'Nymphéas à Giverny',
                font =('Technical', 20), fill ='red')

def tabRenoir(self):
    self.can.create_text(10, 130, anchor =NW,
                text = 'Le moulin de la galette',
                font =('Dom Casual BT', 20), fill ='maroon')

def tabDegas(self):
    self.can.create_text(10, 160, anchor =NW, text = 'Danseuses au repos',
                font =('President', 20), fill ='purple')
```

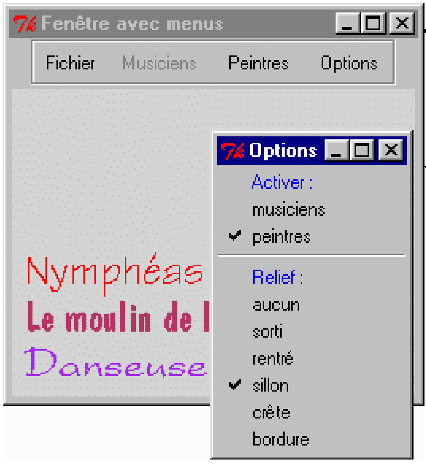
تحليل السكريبت

يمكنك بسهولة صنع قوائم الشلالات، من خلال ربط القوائم الفرعية مع بعضها البعض في مستوى معين (و لكن ينصح أن لا تتجاوز خمسة مستويات متتالية، لأنك ستخسر مستخدمك).

يتم تعريف القائمة الفرعية على أنها قائمة عبيد لقائمة في المستوى السابق (في مثالنا، **me2** تم تعريفها على أنها قائمة "عبيد" لـ **me1**). وسيتم دمجها بمساعدة الأسلوب **add_cascade()**.

واحدة من العناصر معطلة (الخيار **state = DISABLED**). المثال التالي يظهر لك كيف يمكنك تعطيل أو تفعيل عناصر، من خلال البرنامج .

إضافة القائمة Option (خيارات)



تعريف هذه القائمة أكثر تعقيدا، لأننا سوف تصمن استخدام المتغيرات الداخلية لـ tkinter.

وظيفة هذه القائمة هي أكثر تفصيلا : الخيارات المضافة تجعل من الممكن تعطيل أو تفعيل القوائم "Musiciens" و"Peintres" ويمكنك تغيير مظهر شريط القوائم نفسه.

يرجى إذا إضافة الأسطر التالية في منشئ الصنف **MenuBar()** :

```
##### <قائمة خيارات> #####
optMenu = Menubutton(self, text ='Options')
optMenu.pack(side =LEFT, padx ='3')
# متغيرات tkinter :
self.relief = IntVar()
self.actPein = IntVar()
self.actMusi = IntVar()
# جزء "أسفل" القائمة :
self.mo = Menu(optMenu)
self.mo.add_command(label = 'Activer :', foreground ='blue')
self.mo.add_checkbutton(label ='musiciens',
          command = self.choixActifs, variable =self.actMusi)
self.mo.add_checkbutton(label ='peintres',
          command = self.choixActifs, variable =self.actPein)
self.mo.add_separator()
self.mo.add_command(label = 'Relief :', foreground ='blue')
for (v, lab) in [(0,'aucun'), (1,'sorti'), (2,'rentré'),
                 (3,'sillon'), (4,'crête'), (5,'bordure')]:
    self.mo.add_radiobutton(label =lab, variable =self.relief,
                            value =v, command =self.reliefBarre)
# تكامل القائمة :
optMenu.configure(menu = self.mo)
```

... وتعاريف الأساليب التالية دائما للصنف **MenuBar()**:

```
def reliefBarre(self):
    choix = self.relief.get()
```

```
        self.configure(relief =[FLAT,RAISED,SUNKEN,GROOVE,RIDGE,SOLID][choix])

    def choixActifs(self):
        p = self.actPein.get()
        m = self.actMusi.get()
        self.pein.configure(state =[DISABLED, NORMAL][p])
        self.musi.configure(state =[DISABLED, NORMAL][m])
```

قائمة مع خانات اختيار

القائمة الجديدة تتكون من جزئين. ولتسليط الضوء، لقد قمنا بإدراج سطر فاصل و 2 من العناصر الكاذبة (« Activer : » و« Relief : ») التي تخدم العناوين. لقد قمنا بإظهار هذه بلون لا يمكن للمستخدم الخلط مع الأوامر الحقيقية.

تم تجهيز العناصر التالية في الجزء لأول من خانات الاختيار. عندما يقوم المستخدم بالضغط من خلال الفأرة على عنصر أو أكثر من هذه العناصر يتم تفعيل أو تعطيل الخيارات، وهذه الحالة «actif / inactif» يتم عرضها في شكل خانات. التعليمات التي تخدم تنفيذ هذا النوع من القائمة هي ذاتية التوضيح. لديها في الحقيقية هذه العناصر كويدجات من نوع **chekbutton** :

```
    self.mo.add_checkbutton(label = 'musiciens', command = choixActifs,
                                  variable = mbu.me1.music)
```

من المهم أن نفهم هنا أن هذا النوع من الودجة يحتوي بالضرورة متغيرات داخلية، لتخزين حالة «actif / inactif» للودجة. كما قمنا بتفسير هذا أعلاه، هذا المتغير لا يمكن أن يكون متغير بيثون عاديا، لأن أصناف مكتبة tkinter تم كتابتهم بلغات أخرى. وبالتالي لا يمكننا الوصول إلى هذا المتغير الداخلي إلا من خلال كائن-الواجهة، والذي نسميه متغير tkinter لتبسيطه[78].

حتى في مثالنا، استخدمنا صنف **IntVar** tkinter() لصنع كائنات تعادل متغيرات نوع صحيح .

- لقد قمنا بتمثيل هذه الكائنات-المتغيرات، التي قمنا بتخزينها كسمات مثيل: `self.actMusi =IntVar`().
بعد هذه المهمة، كائن مرجع في **self.actMusi** يحتوي على ما يعادل متغير من نوع صحيح، بشكل خاص لـ tkinter.
- ثم يجب ربط خيار المتغير كائن **checkbutton** لمتغير tkinter الذي تم تعريفه :

`self.mo.add_checkbutton(label ='musiciens', variable =self.actMusi)`.

78 انظر أيضا الصفحة 222.

- من الضروري أن تفعل ذلك في خطوتين، لأن tkinter لا يقبل تعيين قيم لمتغيرات بيثون. لسبب بسيط، لا يمكن لبيثون قراءة محتوى متغير tkinter مباشرة. يجب أن تستخدم أساليب خاصة لهذا صنف الكائن : الأسلوب **get()** للقراءة، والأسلوب **set()** للكتابة:219

```
m = self.actMusi.get().
```

في هذه التعليمات، قمنا بتعيين لـ **m** (متغير عادي لبيثون) محتوى المتغير **self.actMusi** tkinter (الذي هو في حد ذاته يرتبط مع تطبيق واضح ومحدد).

كل ما سبق قد يبدو معقدا قليلا. انظر ببساطة إلى أول مواجهة مع مشاكل التواصل بين لغتي برمجة مختلفتين، المستخدمة معا في مشروع مركب .

قائمة مع خيارات حصرية

الجزء الثاني من قائمة "Options" يسمح للمستخدم باختيار شكل من شأنه أن يجعل شريط القوائم، بستة خيارات. وغني عن القول أنه يمكنك تفعيل إحدى هذه الخيارات في كل مرة. لتنفيذ هذا النوع من الوظائف، نقوم كلاسكيا باستدعاء الويدجات من نوع "أزرار راديو". والميزة الأساسية من هذه الويدجات هي أنه يجب أن تكون مرتبطة مع متغير tkinter. كل زر راديو يتوافق مع قيمة معينة، وهذه القيمة تم تعيينها عندما يقوم المستخدم بتحديد الزر.

و هكذا، التعليمة :

```
    self.mo.add_radiobutton(label ='sillon', variable =self.relief,
                            value =3, command =self.reliefBarre)
```

إن تكوين عنصر من قائمة "Options" بحيث يتصرف مثل زر راديو.

عندما يقوم المستخدم بتحديد هذا العنصر، يتم تعيين القيمة 3 إلى المتغير **self.relief** tkinter (ويشار إلى هذا الأخير بمساعدة خيار **variable** للودجة)، ويتم إجراء الاستدعاء نحو أسلوب **reliefBarre()**. وهذا يسترد القيمة المخزنة في المتغير tkinter للقيام بعمله.

في سياق معين من هذه القائمة، أردنا أن نقدم 6 خيارات مختلفة للمستخدم. لذا نحن بحاجة إلى 6 "أزرار راديو"، تمكننا من ترميز 6 تعليمات متماثلة كنا تستنسخها في الأعلى، كل واحدة منها لا تختلف عن الخمسة الأخرى إلا بخياراتها **value** و **label**. في مثل هذا الوضع، الممارسة الجيدة هو وضع قيم نت هذه الخيارات في قائمة، ثم قم باستعراض القائمة باستخدام حلقة **for**، ثم قم بإنشاء مثيل ويدجات مع تعليمة مشتركة :

```
     for (v, lab) in [(0,'aucun'), (1,'sorti'), (2,'rentré'),
                      (3,'sillon'), (4,'crête'), (5,'bordure')]:
         self.mo.add_radiobutton(label =lab, variable =self.relief,
                                 value =v, command =self.reliefBarre)
```

القائمة تستخدم قائمة متكونة من 6 مصفوفات مغلقة Tuples (قيم، ملصقات). في كل واحدة من 6 تكرارات الحلقة، يتم إنشاء مثيل زر راديو جديد، ويتم استخراج الخيارات **label** و **value** من قائمة من خلال المتغيرات **lab** و **v**.

في مشاريعك الخاصة، سوف تجد في كثير من الأحيان أنه يمكنك استبدال تعليمات متماثلة بهيكل برمجي أكثر إحكاما (عادة مزيج من قائمة وحلقة، مثل في المثال أعلاه).

سوف تكتشف شيئا فشيئا تقنيات أخرى لتخفيف كودك : سوف نقدم مثال في الفقرة القادمة. ومع ذلك حاول أن تأخذ في الاعتبار هذه القاعدة : البرنامج الجيد يجب أن يكون دائما قابل للقراءة ومعلق أيضا .

التحكم في تدفق التنفيذ بمساعدة قائمة

يرجى النظر الآن إلى تعريف الأسلوب **reliefBarre()**.

في السطر الأول، الأسلوب **get()** يسمح لنا باسترداد حالة المتغير tkinter الذي يحتوي على رقم الاختيار الذي أدلى به المستخدم في القائمة الفرعية " Relief : ".

في السطر الثاني، استخدمنا محتوى المتغير **choix** لإستخراج قائمة بها 6 عناصر التي نحن مهتمين بها. على سبيل المثال، إذا كان **choix** يحتوي على القيمة 2، سيكون المتغير **SUNKEN** هو المستخدم لإعادة تكوين الودجة.

المتغير **choix** هو المستخدم هنا كمؤشر، ويستخدم لتعيين عنصر قائمة. بدلا من هذا البناء المدمج، ويمكننا برمجة سلسلة من الاختبارات الشرطية، مثل :

```
if choix ==0:
    self.configure(relief =FLAT)
elif choix ==1:
    self.configure(relief =RAISED)
elif choix ==2:
    self.configure(relief =SUNKEN)
...
etc.
```

من وجهة نظر وظيفية، ستكون النتيجة نفسها. يجب الاعتراف أن البناء الذي اخترناه هو أكثر فعالية حيث أن عددا من الخيارات تم إزالته. تخيل على سبيل المثال، أن أحد برامجك الشخصية يجب الاختيار بين عدد كبير من العناصر وهي : بناء من نوع أعلاه، قد تكون هنالك حاجة لترميز صفحات متعدد من **elif**!

و نحن لا نزال نستخدم نفس التقنية في الأسلوب **choixActifs()**. التعليمة:

```
    self.pein.configure(state =[DISABLED, NORMAL][p])
```

يستخدم محتوى المتغير **p** كمؤشر للإشارة إلى أي حالة **DISABLED** و **NORMAL** يجب أن يتم تحديدها لإعادة تكوين القائمة "Peintres" .

عندما يتم استدعاء الأسلوب **choixActifs()** يتم إعادة تكوين قائمتين Peintres و Musiciens لشريط القائمة، لجعلها تبدو "طبيعية" أو "معطلة" وفقا لحالة المتغيرات **m** و **p**، والتي هي في حد ذاتها تعبير عن متغيرات tkinter.

هذه المتغيرات **m** و **p** هي في الواقع لتوضيح السكريبت. سكون من الممكن حذفها، وجعل السكريبت أكثر إحكاما، باستخدام تركيبة من التعليمات. ويمكننا على سبيل المثال استبدال التعليمتين :

```
m = self.actMusi.get()
self.musi.configure(state =[DISABLED, NORMAL][m])
```

بتعليمة واحدة فقط، مثل هذه :

```
self.musi.configure(state =[DISABLED, NORMAL][self.actMusi.get()])
```

نلاحظ أننا يمكننا أن نربح، لكن عندما نختصره نفقد بعض الوضوح .

الضبط المسبق لقائمة

لإنهاء هذا التمرين، يمكنك أن ترى أنك يمكنك أن تضع تحديدات مسبقا، أو تعديل البرنامج.

لذا يرجى قبل إضافة التعليمة التالية في منشئ للصنف **Application()** (فقط قبل التعليمة par، **self.pack()** على سبيل المثال) :

```
mBar.mo.invoke(2)
```

عند تشغيل السكريبت المعدل، يمكنك أن ترى في بداية أن قائمة Musiciens في شريط القوائم مفعلة، في حين أن Peintres ليست كذلك. برمج كما هي، وينبغي أن يكون هذان القائمتين مفعلتين بشكل إفتراضي وهذا في الواقع ما يحدث إذا قمنا بحذف التعليمة

mBar.mo.invoke(2).

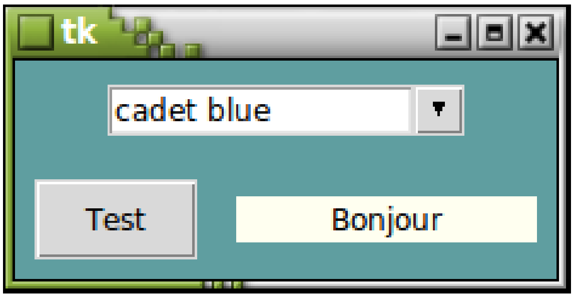
لقد إقترحنا إضافة هذه التعليمة إلى السكريبت لنبين لكم كيف يمكنك تنفيذ نفس العملية من قبل البرنامج مثل التي عادة الضغط من الفأرة.

التعليمة أعلاه تستدعي الودجة **mBar.mo** من خلال تشغيل الأمر المرتبط للعنصر الثاني لهذا الودجة. بالرجوع إلى القائمة، يمكنك التحقق من أن العنصر الثاني هو كائن من نوع **checkbutton** الذي يقوم بتفعيل\تعطيل قائمة Peintres (تذكر مرة أخرى أن الترقيم يبدأ دائما من الصفر).

في بداية البرنامج، كل شيء يحدث كما لو كان المستخدم قام على الفور بالضغط على قائمة Peintres لقائمة Options، مما يؤدي إلى تعطيل القائمة .

تمرين

1.14 أكمل الودجة "مكعب كومبو المبسط" الذي تم وصفه في الصفحة243، بحيث يتم إخفاء قائمة البداية، والزر الصغير على اليمين حقل الإدخال لا يظهر. ما عليك القيام به هو وضع قائمة وشريط تمرير في تافذة "قمر صناعي" بدون حدود (انظر إلى الودجة Toplevel، الصفحة254)، وضعه بشكل صحيح (قد تحتاج إلى الرجوع إلى مواقع التعامل مع tkinter للعثور على المعلومات الضرورية، لكن سيكون هذا من جزء تعلمك !)، وتأكد من أن هذه النافذة تختفي بعد أن يقوم المستخدم بتحديد عنصر في قائمة .



15

تحليل برنامج محدد

في هذا الفصل، سنحاول توضيح عملية تصميم برنامج رسومي، من المسودة الأولى غلى مرحلة متقدمة نسبيا من التطوير. نريد أن نظر مدى قدرة البرمجة الشيئية على تسهيل وتأمين استراتيجية تطوير إضافية التي نريدها[79].

مطلوب استخدام الأصناف، عندما سكون هنالك مشروع جاري نثبت أنه أكثر تعقيدا مما كنا نتخيل أصلا. بالتأكيد سوف تعيش حياة بمسارات مماثلة لتلك التي وصفناها.

لعبة القصف

هذا المشروع[80] استلهم من عمل مماثل أنتجه طلاب السنة النهائية.

من المستحسن البدء بمشروع مثل هذا المشروع من خلال سلسلة من الرسومات الصغيرة والرسوم البيانية، التي تم وصفها في عناصر بناء رسم مختلفة، والتي تسخدم بكثرة. إذا كنت لا ترغب في استخدام التكنولوجيا القديمة ورقة\قلم (المبرهنة حتى الآن)، يمكنك الاستفادة من برامج الرسم التقني، مثل استخدام Draw من المجموعة المكتبية أوبن أوفيس[81] التي كنا قد جعلنا الرسم البياني في الصفحة التالية .

الفكرة بسيطة : لاعبان يتنافسان في برميل. يجب على كل واحد منهم ضبط زاوية إطلاق النار ليحاول الوصول إلى خصمه، والقذائف البالستية تصف المسارات.

---

79 انظر إلى صفحة 8 : البحث عن الأخطاء و التجريب. وأيضا إلى صفحة 261 : نوافذ مع قوائم.

80 نحن لا نتردد هنا في مناقشة تطوير لعبة. لأنه مجال متاح للجميع. و هي أهداف محددة يسهل التعرف إليها . و يمكن تطبيق نفس الأساليب المنتمية إلى تطبيقات أحرى أكثر "جدية" .

81 و هو مجموعة مكتبية كاملة. حرة و مجانية. و هي متوافقة على نطاق واسع مع MS-Office. لينكس و ويندوز و ماك و سولاريس ... و لقد كتب هذا الكتاب بواسطة معالج النصوص الخاص به . يمكنك الحصول عليه عن طريق تحميله من موقع : http://www.openoffice.org

و يتم تعريف مكان وجود المدفع في بداية اللعبة بشكل عشوائي (على الأقل في الأعلى). بعد كل طلقة، يتم استبدال البنادق (لزيادة الإثارة في اللعبة، وبالتالي يتم تعديل الطلقات أكثر صعوبة). يتم تسجيل التسديدات على الهدف.

التصميم الأولي الذي قمنا باستنساخه أعلاه هو واحد من النماذج التي يمكن ان تستغرق عملك التحليلي. قبل البدء في تطوير مشروع برمجة، يجب أن نسعى دائما تفصيل المواصفات. وهذه الدراسة الأولية مهمة للغاية. معظم المبتدئين بدأوا بسرعة بكتابة أسطر عديد من الكود مع فكرة غامضة، لكنه يتجاهل البحث عن الهيكل العام. وقد تصبح برمجتهم خطيرة ثم تصبح فوضى، لأنها ستنفذ هذا الهيكل عاجل أم آجلا. وفي كثير من الأحيان يبدأ بالحذف وإعادة كتابة قطاعات بأكملها من المشروع لأنها صممت بطريقة متجانسة جدا\او تكوينها بشكل غير صحيح .

- متجانسة جدا : هذا قد يعني فشلا في كسر مشكلة معقدة إلى عدة مشاكل فرعية صغيرة أكثر بساطة. على سبيل المثال، تداخل العديد من المستويات من تعليمات مركبة، بدلا من استخدام دالات أو أصناف.
- سوء تكوينه : هذا يعني انه يتعامل فقط مع حالة معينة، بدلا من دراسة الحالة العامة. على سبيل المثال، أعطينا لكائن رسومي أبعادا ثابتة، بدلا من توفير متغيرات للسماح بتغيير حجمه .

عندما تريد أن تطور مشروعا يجب أن تبدأ دائما بمرحلة التحليل، وتنفيذ نتائج هذا التحليل في مجموعة من الوثائق (رسومات وخطط ووصف ...إلخ) التي تشكل المواصفات. للمشاريع الكبيرة، هنالك أيضا أساليب متطورة للتحليل (UML، Merise...إلخ) وهذه لا نستطيع شرحها هنا لأنها تحتاج كتب بأكملها.

يقال، ويجب أن أعترف أنه صعب جدا (و ربما مستحيل) تحليل مشروع برمجي في البداية بأكمله. لأننا عند تشغيله سوف نعرف نقاط ضعفه. وتبقى هنالك حالات استخدام او قيود لم نقصدها أصلا. من جهة أخرى، المشروع البرمجي دائما ما يحتاج إلى التطوير : سوف يحتاج في كثير من الأحيان إلى تعديل المواصفات أثناء التطوير، وليس بالضرورة أنك أخطأت في التحليل الأولي، لكن ببساطة أنك تريد إضافة مميزات إضافية.

و في الختام، حاول دئما التعامل مع مشروع برمجة جديد باحترام النقطتين التاليتين :

- صف مشروعك بالتفصيل قبل البدء بكتابة الأسطر الأولى من التعليمات البرمجية، لتسليط الضوء على المكونات الرئيسية والعلاقة بينها (أعتقد وصف حالات الاستخدام لمستخدم برنامجك).
- عند البدء في العمل الحقيقي، لا تسترسل في كتابة كتل كبيرة من التعليمات. تأكد من تقطيع برنامجك لعدد من المكونات القابلة للتكوين مغلقة بشكل جيد، بحيث يمكنك بسهولة تعديل أي واحد منها دون المساس بتشغيل الأخرين، وحتى إعادة استخدامها في سياقات مختلفة إذا دعت الحاجة إلى ذلك .

و لتلبية هذا الطلب تم اختراع البرمجة الشيئية .

على سبيل المثال انظر إلى المشروع المرسوم في الصفحة السابقة .

قد يميل المبتدئ بإجراء هذه اللعبة باستخدام البرمجة الإجرائية فقط (هذا معناه عدم تحديد أصناف جديدة). وهذا هو أيضا ما قمنا بمعالجته خلال الفصل الأول في الواجهات الرسومية، من الفصل 8. وهذا النهج لا يبرر أن البرامج الصغيرة (تمارين أو اختبارات أولية). عند معالجة مشروع من حجم معين، وتعقيد المشاكل التي نشأت بسرعة سوف تصبح كبيرة جدا، وسيصبح من الضروري تفكيكه وتجزئته.

الأداة البرمجة التي تسمح بهذه التجزئة هي الصنف.

ربما ستفهم أفضل فائدته بمساعدته بالقياس.

جميع الأجهزة الالكترونية تتكون من عدد قليل من المكونات الأساسية، وهي الترانزستورات والثنائيات والمقاومات والمكثفات وإلخ. بنيت الحواسيب الأولى مباشرة من هذه المكونات. وكانت ضخمة ومكلفة وكانت لديها وظائف قليلة جدا ودائما ما تتعطل.

ثم قاموا بتطوير تقنيات جديدة لتغليف مجموعة كبيرة من المكونات الالكترونية الأساسية في علبة. لاستخدام هذه الدوائر الكهربائية المدمجة، لم يعد من الضروري معرفة محتوياته بالضبط : وظيفة واحدة مهمة فقط. كانت الوظائف الأولية المتكاملة لا تزال نسبيا بسيطة : كانت على سبيل المثال، بوابات منطقية، مزازيات ... إلخ. من خلال الجمع بين هذه الدوائر معا، سوف نحصل على المزيد من المميزات المتقدمة، مثل السجلات أو أجهزة فك تشفير، والذي بدوره يجب أن يكون متكامل وهكذا، إلى المعالجات الحالية. وهي تتكون من الملايين من المكونات، ومع ذلك لديها موثوقية عالية.

وفقا لذلك، للإلكترونيات الحديثة التي تريد بناء على سبيل المثال عداد ثنائي (الدارة تتطلب عددا من المقاييس)، فمن الواضح أنه أسهل بكثير وأسرع وأكثر أمانا لاسخدام مقاييس متكاملة، بدلا من خطأ في الجمع بين مئات الترانزستورات والمقاومات.

بطريقة مماثلة، يمكن للمبرمج الحديث الاستفادة من العمل المتراكم من سابقيه في استخدام وظائف الدالات المدمجة في أصناف موجود في بيثون. والأفضل من ذلك، فإنه يمكن بسهولة إنشاء أصناف جديد لتغليف المكونات الرئيسية للتطبيق، وخاصة تلك التي تظهر في نسخ متعددة. وذك أبسط وأسرع وأكثر أمانا من كتل تعليمات تتضاعف في هيئة برامج متماثلة متجانس، ضخمة أكثر ومفهوم أقل.

لدينا الآن المسودة المرسومة. أهم مكونات هذه اللعبة هي البنادق الصغيرة، سوف تكون قادرة على الرسم في مواقع مختلفة وفي إتجاهات مختلفة، ونحن بحاجة على الأقل على نسختين منها.

بدلا من الإستفادة من الرسم قطعة قطعة ولو في أثناء المباراة، نحن مهتمون في بها ككائنات البرامج في حد ذاتها، مع خصائص متعددة وسلوك معين (ما كنا نريد قوله هنا هو أن تكون مجهزة مع آليات مختلفة، يمكننا فعل هذا برمجيا باستخدام أساليب معينة). Il est donc certainement judi cieux de leur consacrer une classe spécifique.

نماذج لصنف مدفع (Canon)

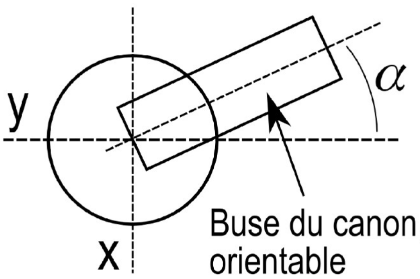
من خلال تعريف هذا الصنف، سوف نفوز في عدة جبهات. ليس فقط جمع كافة التعليمات البرمجية لتصميم وتشغيل المدفع في "كبسولة" واحدة،بعيدا عن بقية البرنامج، لكن بالإضافة إلى ذلك نقدم إماكنية تمثيل أي عدد من المدافع في اللعبة، والذي يتيح لنا المزيد من الفرص للتطوير.

عندما يتم بناء واختبار النموذج الأولي لصنف **Canon**() سيكون من الممكن أيضا تحسينه من خلال منح مميزات إضافية دون تغيير (أو تقليل) واجهته، وهذا يعني على نحو ما "الإرشادات" : بمعرفة التعليمات اللازمة لإنشاء مثيل واستخدامه في مختلف التطبيقات.

سوف ندخل الآن في صلب الموضوع.

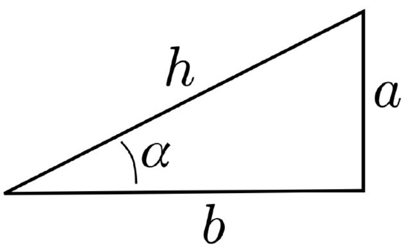
رسم المدفع يمكن تبسيطه إلى أقصى الحدود. شعرنا أننا يمكننا تلخيصها في دائرة إلى جانب مستطيل، وإنه يمكن أيضا أن تعتبر نفسها جزءا بسيطا خط مستقيم سميك بشكل خاص.

إذا تم تعبئتها جميعا بلون واحد (أسود مثلا)، نحصل على نوع من القنابل ذات مصداقية بما فيه الكافية .



في الوسيطة التالية، نحن نفترض أن موقع المدفع هو في الواقع موقع مركز الدائرة (الإحداثيات x و y في الرسم جانبا). هذه النقطة الأساسية تدل أيضا على محور دوران فوهة المدفع، والخط السميك في الذي يمثل الفوهة.

لإنهاء تصميمنا، فلم يتبقَ سوى تحديد إحداثيات الطرف الأخر من الخط. هذه الإحداثيات يمكن حسابها بدون صعوبة كبيرة، تذكر مفهومين أساسيين للمثلث (الجيب والجيب التمام) الذي يجب أن يكون على دراية بها بالتأكيد :



في المثلث القائم، نسبة الجانب المقابل في زاوية ووتر المثلث هي خاصية معينة لهذا الزاوية تسمى جيب الزاوية. جيب الزاوية التمام هي النسبة بين الجانب المجاور للزاوية والوتر .

على سبيل المثال، في الرسم التخطيطي: $\sin A = \frac{a}{h}$ و $\cos A = \frac{b}{h}$.

لتمثيل طرف المدفع، على افتراض أن نعرف طول وزاوية الإطلاق α، لذا يجب علينا رسم خط مستقيم سميك، من إحداثيات مركز الدائرة (x,y) إلى آخر نقطة إلى اليمين فوق، والمسافة الأفقية Δx تساوي l.cos α، والمسافة العمودية Δy تساوي l.sin α.

ملخص كل ما سبق، يجب علينا رسم مدفع في النقطة x,y تتكون ببساطة :

- رسم دائرة سوداء في وسط x, y.
- رسم خط أسود سميك من النقطة x,y إلى النقطة x + l.cos α، y + l.sin α.

يمكننا الآن أن نبدأ في النظر إلى مسودة البرمجة لصنف "Canon". إننا لا نتحدث بعد عن برمجة اللعبة. نريد فقط معرفة ما إذا كان التحليل الذي قمنا به الآن "يأخذ طريقه" عن طريق إجراء نموذج أولي للوظيفة.

النموذج الأولي هو برنامج صغير لاختبار فكرة، التي نقترح دمجها في تطبيق أكبر. بسبب البساطة والإيجاز، بيثون يفسح لك المجال بشكل جيد في تطوير النماذج الأولية، والعديد من المبرمجين يستخدمونها لبرمجة برامج مختلفة ثم سيقومون ربما بإعادة برمجتها بلغات أخرى "ثقيلة"، على سبيل المثال لغة السي.

في نموذجنا الأولي، الصنف **Canon()** لا يملك سوى أسلوبين : المنشئ الذي يقوم بصنع العناصر الأساسية للرسم، والأسلوب الذي يسمح بتعديل ضبط زاوية الإطلاق (الصندوق الخلفي للفوهة). كما فعلنا في كثير من الأحيان في أمثلة أخرى، وسوف نقوم بتضمين بضعة أسطر برمجية في نهاية السكريبت من أجل اختبار الصنف على الفور:

```
from tkinter import *
from math import pi, sin, cos

class Canon(object):
    """Petit canon graphique"""
    def __init__(self, boss, x, y):
        self.boss = boss            # مرجع اللوحة
        self.x1, self.y1 = x, y     # محور دوران المدفع
        # رسم فوهة البندقية, أفقيا للبدء :
        self.lbu = 50               # عرض الفوهة
        self.x2, self.y2 = x + self.lbu, y
        self.buse = boss.create_line(self.x1, self.y1, self.x2, self.y2,
                                     width =10)
        # رسم جسم المدفع أدناه :
        r = 15                      # نصف قطر الدائرة
        boss.create_oval(x-r, y-r, x+r, y+r, fill='blue', width =3)

    def orienter(self, angle):
        "choisir l'angle de tir du canon"
        # يتلقى سلسة نصية <angle> البرامتر.
        # يجب تحويلها إلى عدد حقيقي, ثم إلى راديان :
        self.angle = float(angle)*2*pi/360
        self.x2 = self.x1 + self.lbu*cos(self.angle)
        self.y2 = self.y1 - self.lbu*sin(self.angle)
        self.boss.coords(self.buse, self.x1, self.y1, self.x2, self.y2)

if __name__ == '__main__':
```

```
28#      # كود لتجربة بإختصار الصنف Canon :
29#      f = Tk()
30#      can = Canvas(f,width =250, height =250, bg ='ivory')
31#      can.pack(padx =10, pady =10)
32#      c1 = Canon(can, 50, 200)
33#
34#      s1 =Scale(f, label='hausse', from_=90, to=0, command=c1.orienter)
35#      s1.pack(side=LEFT, pady =5, padx =20)
36#      s1.set(25)                              # زاوية إطلاق النار الأولية
37#
38#      f.mainloop()
```

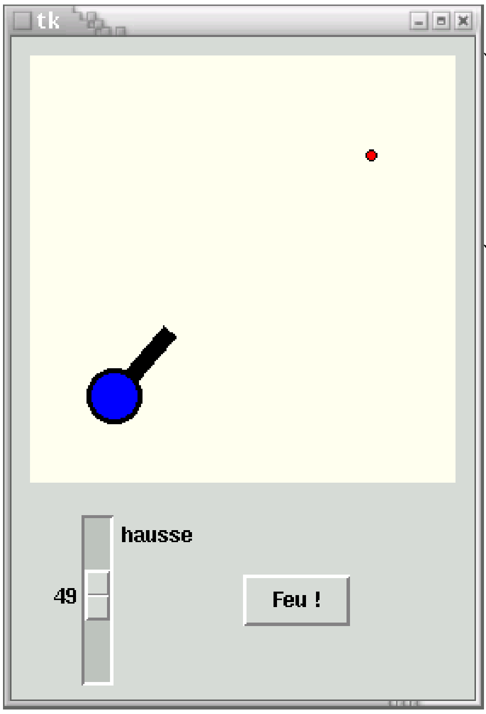
تعليقات

- السطر 6 : في قائمة البرامترات التي سنمررها للمنشئ عند التمثيل، نتوقع أن الإحداثيات x و y، تشير إلى مكان وجود المدفع في اللوحة، ولكن الإشارة إلى اللوحة نفسها (المتغير **boss**). هذا المرجع لا غنى عنه : سوف يستخدم لاستدعاء أساليب اللوحة.

  يمكن أن نشمل أيضا برامتر لاختيار زاوية الإطلاق الأولية، لكن بما أننا نعتزم تنفيذ طريقة محددة لحل هذا التوجه، سيكون من الحكمة استخدامه عند الحاجة.

- الأسطر 7-8 : سوف نستخدم هذه المراجع في جميع الأساليب المختلفة التي نحن نطورها في الصنف. ولذلك يجب علينا أن نصنع سمات مثيل.
- الأسطر من 9 إلى 16 : سوف نصمم الفوهة أولا، ثم جسم المدفع. وهكذا، جزء من الفوهة الظاهرة تبقى مختبئة. وهذا يسمح لنا بتلوين جسم المدفع.
- الأسطر من 18 إلى 25 : هذه الأساليب سيتم استدعاؤها مع البرامتر **angle**، والتي سيتم توفير درجتها(درجة الزاوية) (العد من الأفقي). وهذا يتم عمله بمساعدة الويدجات مثل **Entry** أو **Scale**، وهي سوف تمررها على شكل سلسلة نصية، ونحن سنقوم بتحويلها أولا إلى عدد حقيقي قبل استخدامه في حساباتنا (وقد وصفناها في الصفحة السابقة).
- الأسطر من 27 إلى 38 : لاختبار صنف جديد، سوف نستخدم ودجة **Scale**. لتعريف الموقع الأولي لمؤشره، ثم تعيين زاوية ارتفاع أولية من المدفع، يجب علينا أن نستدعي الأسلوب **set()** (السطر 36).

إضافة الأساليب للنموذج



نموذجنا هو وظيفي، ولكنها بدائية إلى حد كبير. ونحن الآن بحاجة إلى تطوير القدرة على إضافته إلى إطلاق النار.

سيتم التعامل مع هذه كرصاصة من شأنها أن تكون دوائر بسيطة من فوهة المدفع مع سرعة أولية مماثلة للفوهة. لجعل تتبعها واقعيا، يجب أن نتذكر بعض العناصر الفيزيائية .

مثل الكائن يطرح ويترك نفسه ليتطور في الفضاء، إذا أهملنا الظاهرة الثانوية مثل مقاومة الهواء .

مثل الكائن يطرح ويترك نفسه ليتطور في الفضاء، إذا أهملنا الظاهرة الثانوية مثل مقاومة الهواء.

هذه المشكلة قد تبدو معقدة، ولكنها في الواقع بسيطة جدا : إن مجرد الاعتراف بأن الكرة تتحرك أفقيا وعموديا، وأن هاتان الحركتان على الرغم من أنها متزامنة، فهي مستقلة عن بعضها البعض.

سوف نقوم الآن بإنشاء حلقة الرسوم المتحركة، التي من خلالها يمكنك إعادة حساب الإحداثيات **x** و **y** الجديدة وللكرة على فترات منتظمة من الزمن، مع العلم أن :

- الحركة أفقية موحدة. في كل تكرار، فقط زيادة التدريجية في إحداثية **x** للكرة، وأن نقوم دائما بإضافة نفس مكان Δx .
- التسارع بشكل موحد والحركة الأفقية، هذا يعني ببساطة أن كل تكرار، يجب إضافة الإحداثيات **y** بإزاحة Δy نفسها مما يزيد تدريجيا، ودائما في نفس المقدار.

انظر الآن إلى هذا السكريبت :

لنبدأ، يجب أن تضيف الأسطر التالية إلى نهاية أسلوب المنشئ. وسوف يتم استخدامه لصنع الكائن "القذيفة". وإعداد متغير المثيل من شأنه أن يسخدم رسوم متحركة. يتم صنع القذيفة مع الحد الأدنى للأبعاد (دائرة بكسل واحد) وأن تظل تقريبا غير مرئية :

```
        # رسم قذيفة (تم إختصارها إلى نقطة واحدة, قبل الرسوم المتحركة) :
        self.obus =boss.create_oval(x, y, x, y, fill='red')
        self.anim =False                        # التبديل إلى الرسوم المتحركة
        # إيجاد عرض وارتفاع اللوحة :
        self.xMax =int(boss.cget('width'))
        self.yMax =int(boss.cget('height'))
```

و السطران الأخيران يستخدمان الأسلوب **cget()** للودجة "السيد" (اللوحة، هنا)، من أجل العثور بعض من خصائصه. نريد أن يكون صنفنا **Canon** عاما، وهذا يعني أنه قابل لإعادة الاستخدام في أي سياق، ونحن بالتالي لا يمكننا الاعتماد على الأبعاد المحددة للوحة الذي سيتم استخدامه للمدفع .

> يقوم *tkinter* بإرجاع هذه القيم كسلاسل نصية. ولذلك من الضروري تحويلها إلى نوع رقمي إذا كنا نريد استخدامها في عملية حسابية .

ثم نحن بحاجة إلى إضافة أسلوبين جديدين : واحد لإطلاق النار، والأخر لإدارة الرسوم المتحركة للكرة بعد إطلاقها :

```
1#      def feu(self):
2#          "déclencher le tir d'un obus"
3#          if not self.anim:
4#              self.anim =True
5#              # مكان إطلاق القذيفة (الفوهة) :
6#              self.boss.coords(self.obus, self.x2 -3, self.y2 -3,
7#                                          self.x2 +3, self.y2 +3)
8#              v =15              # السرعة الأولية
9#              # المكونات الأفقية والعمودية لهذه السرعة :
10#             self.vy = -v *sin(self.angle)
11#             self.vx = v *cos(self.angle)
12#             self.animer_obus()
13#
14#     def animer_obus(self):
15#         "animation de l'obus (trajectoire balistique)"
16#         if self.anim:
17#             self.boss.move(self.obus, int(self.vx), int(self.vy))
18#             c = tuple(self.boss.coords(self.obus))      # الإحداثيات الناتجة
19#             xo, yo = c[0] +3, c[1] +3    # إحداثيات مركز القذيفة
20#             if yo > self.yMax or xo > self.xMax:
21#                 self.anim =False         # إيقاف التحريك
22#             self.vy += .5
23#             self.boss.after(30, self.animer_obus)
```

تعليقات

- الأسطر من 1 إلى 4 : هذا الأسلوب سيتم استدعاؤه بالضغط على الزر. فإنه سوف يتسبب في تحريك القذيفة، وتعيين قيمه "الحقيقية" إلى "رسوم متحركة" (المتغير **self.anim** : انظر أدناه)؟ ومع ذلك، يجب علينا أن نضمن مدة لهذه الرسوم المتحركة، ضغطة جديدة على زر لا يمكنها تنشيط حلقات أخرى لتحريك الرسوم. وهذا هو دور السطر الثالث : كتلة التعليمات التي تعمل إذا كان المتغير **self.anim** قيمته "faux"، مما يعني أن الرسوم المتحركة لن تبدأ بعد.
- الأسطر من 5 إلى 7 : لوحة tkinter لديه أسلوبان لنقل الكائنات الرسومية:

-الأسلوب **coords()** (المستخدم في السطر السادس) ينفذ الموضع المطبق، ومع ذلك يجب أن يتم توفير جميع المعلومات للكائن (كما لو أننا أعدنا رسمه) .

-الأسلوب **move()** (المستخدم في وقت لاحق، السطر 17)، يؤدي إلى النزوح نسبيا، وهي تستخدم مع برامترين فقط، هما المكونان الأفقي والعمودي للحركة المطلوبة .

• الأسطر من 8 إلى 12 : يتم اختيار السرعة الأولية للطلقة في السطر 9. كما قمنا نحن بشرحه في الصفحة السابقة، حركة الكرة هي نتيجة لحركة أفقية ولحركة عمودية. نحن نعرف قيمة السرعة الأولية والميل (و هذا يعني زاوية إطلاق النار). لتحديد المكونات الأفقية والعمودية لهذه السرعة، نحن سنقوم فقط باستخدام العلاقات المثلثية المماثلة لتلك التي استخدمت بالفعل في رسم فوهة المدفع. توقيع - المستخدم في السطر 10 يات من إحداثيات العمودية من الأعلى إلى الأسفل.

السطر 12 يفعل (ينشط) الحركة نفسها .

• الأسطر من 14 إلى 23 : هذا الإجراء يقوم باستدعاء نفسه كل 30 ميللي ثانية عن طريق الأسلوب **after()** الذي تم استدعاؤها في السطر 23. وهذه تكمل لطالما المتغير **self.anim** (محرك الرسوم المتحركة الخاص بنا) يبقى "صحيحا - vraie"،و حالة تتغير عند إحداثيات القذيفة تخرج من اختبار حدود (اختبار في السطر 20).

• الأسطر 18-19 : لمعرفة الإحداثيات بعد كل إزاحة، نقوم في كل مرة باستدعاء الأسلوب **coords()** للوحة : يستخدم هذه المرة مع مرجع الكائن الرسومي كبرامتر واحد، سوف تقوم بإرجاع أربعة إحداثيات في كائن itérable التي يمكن تحويلها إلى قائمة أو إلى مصفوفة مغلقة Tuple بمساعدة الدالات المدمجة **list()** و **tuple()**.

• الأسطر 17 - 22 : الإحداثيات الأفقية للطلقة تزيد دائما بمقدار(حركة موحدة)، في حين أن زيادات التنسيق العمودي من قبل المقدار الذي هو في حد ذاته في كل مرة في السطر 24 (الحركة الموحدة بشكل تسارعي). والنتيجة هي مسار مكافئ .

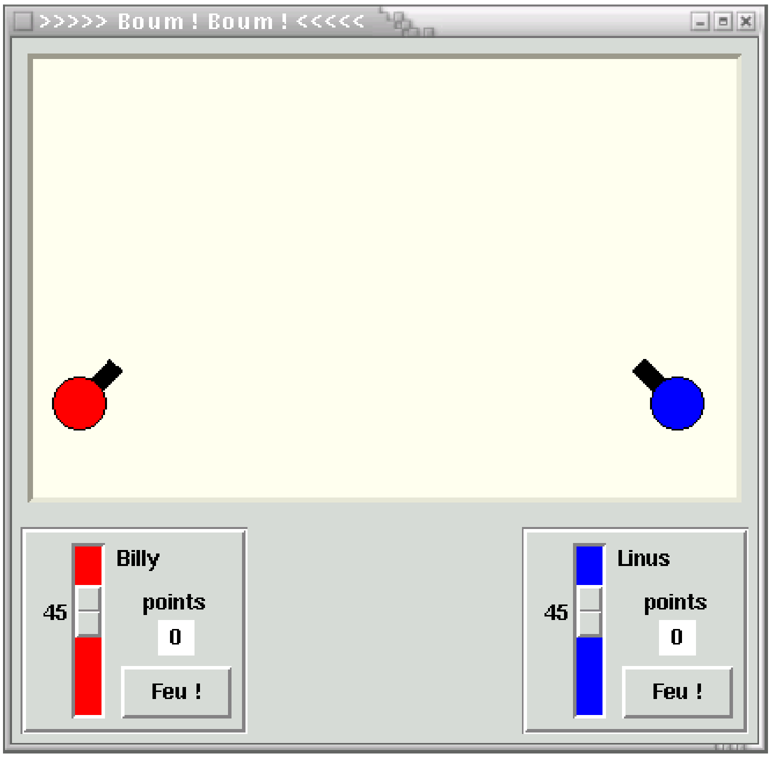
> المعامل += سيمح بزيادة إلى المتغير:
>
> *a += 3* تعادل *a = a + 3*. لاحظ أن استخدام هذا العامل الخاص هو الأكثر كفاءة من إعادة التخصيص التي استخدمناها حتى الآن
>
> بداية من الإصدار 2.3. يقوم بيثون تلقائيا بتهيئة متغيرين اسمهما ***True*** و ***False*** لتمثيل الصحيح والخطأ تعبير (لاحظ أن الاسمين يبدآن بحرف كبير). كما فعلنا في السكريبت أعلاه. يمكنك استخدام هذه المتغيرات في تعابير شرطية لزيادة سهولة قراءة كودك. فإذا كنت تفضل. يمكنك استخدام القيم الرقمية' كما فعلنا سابقا (انظر إلى "صحة\خطأ تعبير" صفحة 56).

و أخيرا، فإنه لا يزال إضافة زر الزناد في النافذة الرئيسية. سطر مثل هذا (سوف يتم وضعها في تعليمات الاختبار البرمجية تقوم بشكل جيد :

```
Button(f, text='Feu !', command =c1.feu).pack(side=LEFT)
```

تطوير تطبيق

الآن لدينا صنف الكائنات "canon" assez bien dégrossie ونرى أن تطوير التطبيق نفسه. ومنذ أن قررنا استخدام منهجية البرمجة الشيئية، يجب علينا تطوير هذا التطبيق على أنه مجموعة من الكائنات التي تتفاعل من خلال أساليبها .



العديد من هذه الكائنات تأتي من أصناف قائمة، بطبيعة الحال : لوحة، أزرار ...إلخ. لكن رأينا في الصفحات السابقة لدينا اهتمام بتجميع مجموعات محددة جيدا من هذه الكائنات الأساليب في أصناف جديدة، كل مرة يمكننا تحديد هذه المجموعات لميزة معينة. على سبيل المثال هذه مجموعة من الدوائر والخطوط المتحركة، من استدعاء "canon - المدفع".

هل يمكننا أن نجعل لمشروعنا الأولي مكونات أخرى ينبغي أن يتم تغليفها في أصناف جديدة ؟ بالتأكيد نعم، يوجد على سبيل المثال لوحة التحكم يمكننا أن نربطها مع كل مدفع : يمكننا جمع ما يصل إلى الأعلى (زاوية إطلاق النار)، زر إطلاق النار، نقاط المتحصل عليها، ومؤشرات أخرة ربما لاتزل مثل اسم اللاعب. ومن المثير للإهتمام بشكل خاص لصنف معينة، ونحن نعلم منذ البداية أننا نأخذ مثيلين.

و هنالك أيضا التطبيق نفسه، بطبيعة الحال. عن طريق التغليف في صنف، وسوف نبذل هدفنا الرئيسي الذي يؤدي لكل الأخرين.

يرجى الآن تحليل السكريبت أدناه. سوف تجد الصنف **Canon()** مطور : أضفنا بعض السمات الإضافية وثلاثة أساليب إضافية، من أجل إدارة حركة المدفع، فضلا عن الأهداف.

الصنف **Application()** يقوم باستبدال كود اختبار النماذج أعلاه. نحن سوف نقوم بتمثيل كائنين **Canon()**، وكائنين لصنف جديد **Pupitre()**ـ الذي وضعناه في قاموس التنبؤ بالتطورات المستقبلية (بمكننا أن نتصور فعلا زيادة عدد المدافع وبالتالي عدد المناضد). اللعب الآن وظيفية : المدافع تتحرك بعد كل حلقة، ويتم تسجيل الأهداف .

```
from tkinter import *
from math import sin, cos, pi
from random import randrange

class Canon(object):
    """Petit canon graphique"""
    def __init__(self, boss, id, x, y, sens, coul):
        self.boss = boss             # مرجع اللوحة
        self.appli = boss.master     # مرجع نافذة التطبيق
        self.id = id                 # تعريف مدفع (سلسلة)
        self.coul = coul             # اللون المرتبط بالمدفع
        self.x1, self.y1 = x, y      # محور دوران المدفع
        self.sens = sens             # اتجاه الإطلاق (-1:يسار, +1:يمين)
        self.lbu = 30                # طول الفوهة
        self.angle = 0               # الزيادة الافتراضية (زاوية الإطلاق)
        # إيجاد عرض وارتفاع اللوحة :
        self.xMax = int(boss.cget('width'))
        self.yMax = int(boss.cget('height'))
        # رسم فوهة المدفع (أفقي) :
        self.x2, self.y2 = x + self.lbu * sens, y
        self.buse = boss.create_line(self.x1, self.y1,
                                     self.x2, self.y2, width =10)
        # رسم جسم المدفع (دائرة ملونة) :
        self.rc = 15                 # نصف قطر الدائرة
        self.corps = boss.create_oval(x -self.rc, y -self.rc, x +self.rc,
                                      y +self.rc, fill =coul)
        # رسم قذيفة مخفية (نقطة خارج اللوحة) :
        self.obus = boss.create_oval(-10, -10, -10, -10, fill='red')
        self.anim = False            # مؤشرات الرسوم المتحركة
        self.explo = False           #    والإنفجار

    def orienter(self, angle):
        "régler la hausse du canon"
        # يتلقى البرامتر <angle> كسلسلة.
        # يجب تحويلها إلى عدد حقيقي, ثم إلى راديان :
        self.angle = float(angle)*pi/180
        # استخدام الأسلوب coords الذي يفضل مع أعداد صحيحة :
        self.x2 = int(self.x1 + self.lbu * cos(self.angle) * self.sens)
        self.y2 = int(self.y1 - self.lbu * sin(self.angle))
        self.boss.coords(self.buse, self.x1, self.y1, self.x2, self.y2)

    def deplacer(self, x, y):
        "amener le canon dans une nouvelle position x, y"
        dx, dy = x -self.x1, y -self.y1      # قيمة الإزاحة
        self.boss.move(self.buse, dx, dy)
        self.boss.move(self.corps, dx, dy)
        self.x1 += dx
        self.y1 += dy
        self.x2 += dx
```

```
        self.y2 += dy

    def feu(self):
        "tir d'un obus - seulement si le précédent a fini son vol"
        if not (self.anim or self.explo):
            self.anim =True
            # جلب وصف جميع المدافع الحالية :
            self.guns = self.appli.dictionnaireCanons()
            # موقع بدء القذيفة (فوهة المدفع) :
            self.boss.coords(self.obus, self.x2 -3, self.y2 -3,
                                        self.x2 +3, self.y2 +3)
            v = 17                    # السرعة الأولية
            # المكونات الأفقية والعمودية لهذه السرعة :
            self.vy = -v *sin(self.angle)
            self.vx = v *cos(self.angle) *self.sens
            self.animer_obus()
            return True       # => إشارة إلى أن القذيفة أطلقت
        else:
            return False      # => لم يتم إطلاق القذيفة

    def animer_obus(self):
        "animer l'obus (trajectoire balistique)"
        if self.anim:
            self.boss.move(self.obus, int(self.vx), int(self.vy))
            c = tuple(self.boss.coords(self.obus))     # الإحداثيات الناتجة
            xo, yo = c[0] +3, c[1] +3      # إحداثيات مركز القذيفة
            self.test_obstacle(xo, yo)     # a-t-on atteint un obstacle ?
            self.vy += .4                  # التسارع العمودي
            self.boss.after(20, self.animer_obus)
        else:
            # الرسوم المتحركة إنتهت – إخفاء القذائف ونقل المدافع :
            self.fin_animation()

    def test_obstacle(self, xo, yo):
        "évaluer si l'obus a atteint une cible ou les limites du jeu"
        if yo >self.yMax or xo <0 or xo >self.xMax:
            self.anim =False
            return
        # analyser le dictionnaire des canons pour voir si les coord.
        # de l'un d'entre eux sont proches de celles de l'obus :
        for id in self.guns:              # id = المفتاح في القاموس.
            gun = self.guns[id]           # القيم المطابقة
            if xo < gun.x1 +self.rc and xo > gun.x1 -self.rc \
            and yo < gun.y1 +self.rc and yo > gun.y1 -self.rc :
                self.anim =False
                # رسم انفجار القذيفة (دائرة صفراء) :
                self.explo = self.boss.create_oval(xo -12, yo -12,
                             xo +12, yo +12, fill ='yellow', width =0)
                self.hit =id       # مرجع إصابة الهدف
                self.boss.after(150, self.fin_explosion)
                break

    def fin_explosion(self):
        "effacer l'explosion ; réinitaliser l'obus ; gérer le score"
        self.boss.delete(self.explo)    # مسح الإنفجار
        self.explo =False               # إذن لإطلاق نار جديد
        # إشارة نجاح إلى النافذة الرئيسية :
        self.appli.goal(self.id, self.hit)

    def fin_animation(self):
        "actions à accomplir lorsque l'obus a terminé sa trajectoire"
```

```
        self.appli.disperser()          # نقل المدافع
        # إخفاء القذيفة (بإرسالها خارج اللوحة) :
        self.boss.coords(self.obus, -10, -10, -10, -10)

class Pupitre(Frame):
    """Pupitre de pointage associé à un canon"""
    def __init__(self, boss, canon):
        Frame.__init__(self, bd =3, relief =GROOVE)
        self.score =0
        self.appli =boss                # مرجع التطبيق
        self.canon =canon               # مرجع المدفع المرتبط
        # نظام تحكم في زاوية الإطلاق :
        self.regl =Scale(self, from_ =85, to =-15, troughcolor=canon.coul,
                         command =self.orienter)
        self.regl.set(45)               # الزاوية الأولية للإطلاق
        self.regl.pack(side =LEFT)
        # علامة تعريف المدفع :
        Label(self, text =canon.id).pack(side =TOP, anchor =W, pady =5)
        # زر الإطلاق :
        self.bTir =Button(self, text ='Feu !', command =self.tirer)
        self.bTir.pack(side =BOTTOM, padx =5, pady =5)
        Label(self, text ="points").pack()
        self.points =Label(self, text=' 0 ', bg ='white')
        self.points.pack()
        # وضع على يمين أو اليسار اعتمادا على اتجاه المدفع :
        if canon.sens == -1:
            self.pack(padx =5, pady =5, side =RIGHT)
        else:
            self.pack(padx =5, pady =5, side =LEFT)

    def tirer(self):
        "déclencher le tir du canon associé"
        self.canon.feu()

    def orienter(self, angle):
        "ajuster la hausse du canon associé"
        self.canon.orienter(angle)

    def attribuerPoint(self, p):
        "incrémenter ou décrémenter le score, de <p> points"
        self.score += p
        self.points.config(text = ' %s ' % self.score)

class Application(Frame):
    '''Fenêtre principale de l'application'''
    def __init__(self):
        Frame.__init__(self)
        self.master.title('>>>>> Boum ! Boum ! <<<<<')
        self.pack()
        self.jeu = Canvas(self, width =400, height =250, bg ='ivory',
                          bd =3, relief =SUNKEN)
        self.jeu.pack(padx =8, pady =8, side =TOP)

        self.guns ={}           # قاموس المدفع الموجودة
        self.pupi ={}           # قاموس طاولات اللعب
        # تمثيل كائني مدفع (+1, -1 = معاكس) :
        self.guns["Billy"] = Canon(self.jeu, "Billy", 30, 200, 1, "red")
        self.guns["Linus"] = Canon(self.jeu, "Linus", 370,200,-1, "blue")
        # تمثيل طاولتي تسجيل مرتبطة بهذه المدافع :
        self.pupi["Billy"] = Pupitre(self, self.guns["Billy"])
        self.pupi["Linus"] = Pupitre(self, self.guns["Linus"])
```

```
172#
173#     def disperser(self):
174#         "déplacer aléatoirement les canons"
175#         for id in self.guns:
176#             gun =self.guns[id]
177#             # وضع على يمين أو اليسار اعتمادا على اتجاه المدفع :
178#             if gun.sens == -1 :
179#                 x = randrange(320,380)
180#             else:
181#                 x = randrange(20,80)
182#             # النزول :
183#             gun.deplacer(x, randrange(150,240))
184#
185#     def goal(self, i, j):
186#         "le canon <i> signale qu'il a atteint l'adversaire <j>"
187#         if i != j:
188#             self.pupi[i].attribuerPoint(1)
189#         else:
190#             self.pupi[i].attribuerPoint(-1)
191#
192#     def dictionnaireCanons(self):
193#         "renvoyer le dictionnaire décrivant les canons présents"
194#         return self.guns
195#
196# if __name__ =='__main__':
197#     Application().mainloop()
```

تعليقات

- السطر 7 : وبالمقارنة مع النموذج، تم إضافة 3 برامترات إلى الأسلوب المنشئ. البرامتر **id** يسمح لنا بتعريف كل مثيل في الصنف **Canon()** بمساعدة أي اسم. والبرامتر **sens** تشير إلى اتجاه البندقية فإذا كانت على اليمين (sens = 1) وإذا كانت على اليسار (sens = -1). البرامتر **coul** الخاص بلون المرتبط بالمدفع.
- السطر 9 : جميع ويدجات tkinter يمكنها الوصول إلى سمة **master** التي تحتوي على مرجع ودجتهم "السيد" المحتمل (الحاوي). هذا المرجع هو بالنسبة لنا التطبيق الرئيسي لقد قمنا بتنفيذ تقنية مماثلة لمرجع اللوحة، ذلك باستخدام سمة **boss**.
- الأسطر من 42 إلى 50 : هذا الأسلوب يسمح لنا بوضع المدفع في مكان جديد. استخدمه لإعادة وضع المدفع بشكل عشوائي بعد كل طلقة، مما يزيد من الاهتمام في اللعبة.
- الأسطر 56 و 57 : نحن نحاول بناء صنف المدفع بحيث يمكن استخدامه في مشاريع أكبر، تشمل على أي عدد من أصناف المدافع التي يمكنها الظهور والإختفاء في وسط القتال. في هذا المنظور، لابد أن لدينا وصف لجميع المدافع الحالية، قبل كل طلقة، بحيث يمكن تحديد ما إذا كان قد تم إصابة الهدف أو لا. وهذا الوصف يتم صنعه بواسطة التطبيق الرئيسي، في قاموس، والتي يمكن طلب نسخة من خلال الأسلوب **dictionnaireCanons()**.

- الأسطر من 66 إلى 68 : في هذا المنظور العام نفسه، قد يكون من المفيد أن أبلغ أن البرنامج الاستدعاء أطلق النار فعلا أو لا.
- السطر 76 : يتم التعامل مع رسوم المتحركة للطلقة (أو القذيفة) من خلال أسلوبين متكاملة. لتوضيح الكود، وضعنا في أسلوب منفصل تعليمات تحدد طريقة ما إذا تم التوصل إلى الهدف (الأسلوب **test_obstacle()**).
- الأسطر من 79 إلى 81 : لقد رأينا سابقا أن الرسوم المتحركة للقذيفة تتوقف عند تعيين القيمة "سالب - fausse" للمتغير **self.anim**. الأسلوب **animer_obus()** يوقف الحلقة قم ينفذ المود في السطر 81.
- الأسطر 83 إلى 100 : هذا الأسلوب يقيم ما إذا كانت الإحداثيات الحالية بطلقة خارجة من حدود النافذة، أو إنها تقترب من قذيفة أخرى. في كلا الحالتين، يتم تفعيل مبدل الرسوم المتحركة، لكن في الحالة الثانية، نرسم "انفجار" أصفر، ويتم تخزين مرجع المدفع. ينم استدعاء أسلوب المرفق **fin_explosion()** بعد وقت قصر لإنهاء العمل، وهذا معناه حذف دائرة الانفجار وإرسال رسالة إلى نافذة الرئيسية للإشارة إلى أنه ضرب.
- الأسطر 115 إلى 152 : ستم تعريف الصنف **Pupitre()** في ودجة جديد مشتق من الصنف **Frame()**، وهي تقنية أصبحت الآن مألوفة وهذا الودجة الجديد يجمع أوامر الارتفاع والإطلاق النار ، ثم يعرض النقاط المرتبطة بالمدفع المحددة جيدا. ويتم توفير مراسلات بصرية بين الإثنين من خلال اعتماد لون مشترط. الأساليب **tirer()** ، **orienter()** تتواصل مع الكائن **Canon()** المرتبط بها، عن طريق أساليبها.
- الأسطر من 154 إلى 171 : نافذة التطبيق هي أيضا ودجة مشتق من **Frame()**. منشئه يمثل مدفعين ويشير إلى مواقع الإطلاق، وتم وضع هذه الكائنات في قاموسين **self.guns** و **self.pupi**. هذا يسمح بتنفيذ معالجات مختلفة على منهجية كل واحد منهم (على سبيل المثال الأسلوب التالي). بالقيام بذلك، فإنه تحتفظ أيضا إمكانية زيادة عدد المدافع إذا لزم الأمر، في تطويرات لاحقة من البرنامج.
- * الأسطر من 173 إلى 183 : يتم استدعاء هذه الأساليب بعد كل طلقة لنقل المدفعين بشكل عشوائي، مما يزيد من صعوبة اللعبة .

تطويرات إضافية

كما هو موضح أعلاه، برنامجنا هو أكثر أو أقل من الموصفات الأصلية، لكن من الواضح أننا نتمكن من الاستمرار في تحسينه.

أ) ينبغي لنا أن نضع مثالا أفضل. ما معنا هذا. في شكله الحالي، لعبتنا لديها حجم لوحة محدد سابقا (400 × 250 بيكسل، انظر للسطر 161). فإذا أردنا تغيير هذه القيم، فإننا نحتاج أيضا إلى ضمان تعديل أسطر أخرى من السكريبت حيث الأبعاد المعنية (على سبيل المثال هذه الأسطر 168-169 أو 179-184). قد تصبح هذه الأسطر مترابطة إذا أضفنا العديد من الميزات

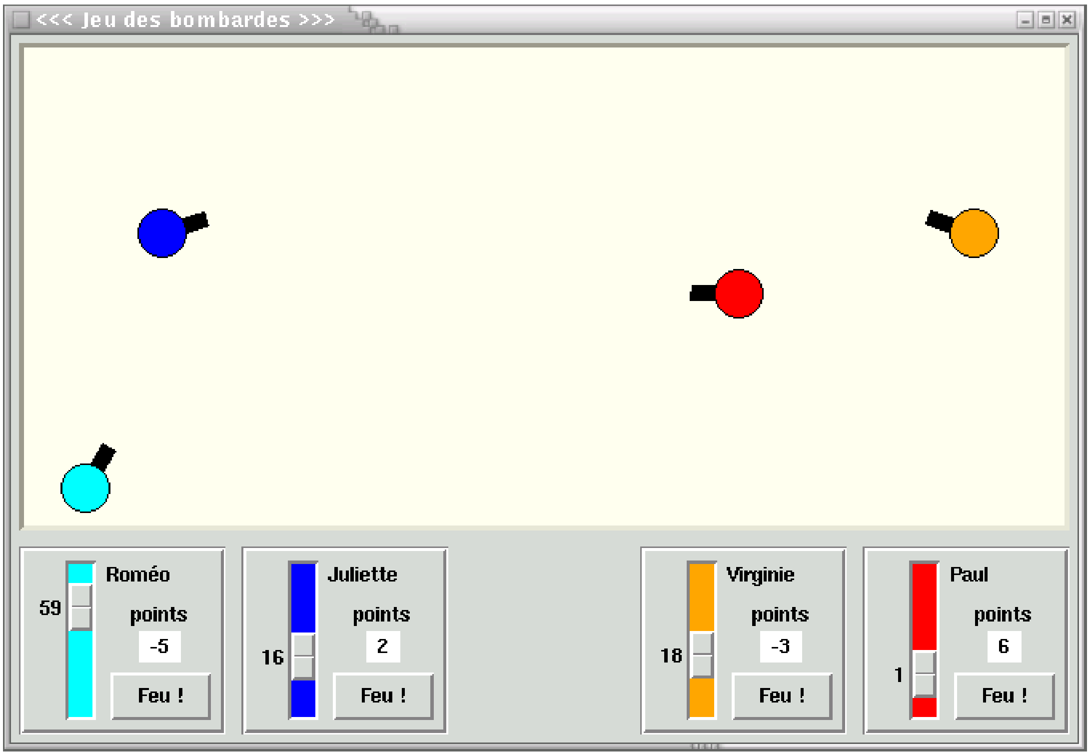
الأخرى. سيكون من الحكمة تغيير حجم اللوحة بمساعدة متغيرات، قيمته تم تعريفها في مكان واحد. هذه المتغيرات سيتم استخدامهم في جميع الأسطر التعليمات التي تشترك فيها أبعاد اللوحة.

لقد قمنا بالفعل بجزء من هذا العمل : في الصنف **Canon()**ـ في الحقيقة أبعاد اللوحة سيتم إستردادها باستخدام أسلوب محدد مسبقا (انظر للسطور 17-18)، ووضعها في سمات المثيل التي يمكن استخدامها في أي في صنف.

ب) بعد كل طلقة، سوف نقوم بنقل المدافع عشوائيا ، وإعادة تعريف إحداثياتها. ربما يكون أكثر واقعية نسبيا والسبب الحقيقي في النزوح، بدلا من إعادة تعريف المواقع المطلقة عشوائيا. للقيام بذلك، يكفي أن تعيد عمل الأسلوب **deplacer()** للصنف **Canon()**. في الحقيقة، سيكون أكثر إثارة للاهتمام جعل هذه الطريقة يمكن أن تنتج ذلك، فضلا عن نزوح تحديد المواقع النسبية المطبقة، بناءا على القيمة الممررة كبرامتر.

ج) ينبغي تحسين نظام التحكم في إطلاق النار : لأن لدينا فقط نظام واحد وهو الفأرة، إرسال اللاعبين بالتناوب، وليس لدينا آلية لإجبارهم على القيام بذلك. لذا اعتمد على النهج الذي يوفر أوامر الارتفاع وإطلاق النار حتى باستخدام بعض مفاتيح لوحة المفاتيح التي يجب أن تختلف بين كلا اللاعبين.

د) ولكن الأكثر إثارة للاهتمام في تطوير برنامجنا لجعله برنامج يعمل على الشبكة. اللعبة سيتم تثبيتها على مجموعة من الأجهزة المتعددة التي تتواصل مع كل لاعب للتحكم على مدفع واحد. سيكون أكثر جاذبية السماح بتنفيذ أكثر من مدفعين، للسماح بالقتال التي تشمل على الكثير من اللاعبين .



و هذا النوع من التطوير، يتطلب منا إتقان مجالين من المجالات التي هي خارج إطار الدورة :

- تقنية sockets، التي تسمح بالاتصال بين جهازي حاسوب.
- تقنية threads، التي تسمح لبرنامج واحد بتنفيذ عدة مهام في وقت واحد (و هذا ضروري، إذا كنت تريد بناء تطبيق يمكنه التواصل مع عدة شركاء) .

هذه المواد ليست جزءاً من الكائنات التي وضعناها لهذه الدورة، والتي تشمل معالجة وحده فصلا كاملا. نحن لا نناقش هذه المسألة هنا. للمهتمين بهذا الموضوع : هذا الفصل موجود، ولكنه كتكملة لنهاية الكتاب (الفصل 19) : وسوف تجد نسخة من لعبتنا تعمل على الشبكة.

و في الوقت نفسه، لا تزال ترى كيف يمكننا إحراز المزيد من التقدم في تحقيق بعض التحسينات في مشروعنا التي من شأنها أن تجعل اللعبة لأربعة لاعبين. وسوف نقوم أيضا بوضع برمجتنا مقسمة بشكل جيد، بحيث أن أساليب الأصناف قابلة لإعادة الاستخدام. وسنرى أيضا كيف يمكننا تغيير الطريقة، دون المساس (التغيير) بالتعليمات البرمجية الموجودة، وسنقوم بهذا عن طريق الميراث لصنع أصناف جديدة من تلك المكتوبة.

نبدأ بحفظ عملنا السابق في ملف، (و الذي نفترض أن له بقية) واسم الملف هو: **canon03.py**.

لدينا الآن وحدة بيثون حقيقية، والتي يمكننا استدعاؤها في سكريبت جديد بمساعدة تعليمة واحدة (في سطر واحد). مـن خلال استغلال هذه التقنية، سوف نواصل تحسين طلبنا، عن طريق الحفاظ على أعيننا الجديد :

```
from tkinter import *
from math import sin, cos, pi
from random import randrange
import canon03

class Canon(canon03.Canon):
    """Canon amélioré"""
    def __init__(self, boss, id, x, y, sens, coul):
        canon03.Canon.__init__(self, boss, id, x, y, sens, coul)

    def deplacer(self, x, y, rel =False):
        "déplacement, relatif si <rel> est vrai, absolu si <rel> est faux"
        if rel:
            dx, dy = x, y
        else:
            dx, dy = x -self.x1, y -self.y1
        # الحدود الأفقية :
        if self.sens ==1:
            xa, xb = 20, int(self.xMax *.33)
        else:
            xa, xb = int(self.xMax *.66), self.xMax -20
        # لا تتحرك إلا داخل الحدود :
        if self.x1 +dx < xa:
            dx = xa -self.x1
        elif self.x1 +dx > xb:
            dx = xb -self.x1
        # الحدود العمودية :
        ya, yb = int(self.yMax *.4), self.yMax -20
        # لا تتحرك إلا داخل الحدود :
        if self.y1 +dy < ya:
            dy = ya -self.y1
        elif self.y1 +dy > yb:
            dy = yb -self.y1
        # تحريك فوهة وجسم المدفع :
        self.boss.move(self.buse, dx, dy)
        self.boss.move(self.corps, dx, dy)
        # renvoyer les nouvelles coord. au programme appelant :
        self.x1 += dx
        self.y1 += dy
        self.x2 += dx
        self.y2 += dy
        return self.x1, self.y1

    def fin_animation(self):
        "actions à accomplir lorsque l'obus a terminé sa trajectoire"
        # تحريك المدفع الذي سيطلق النار :
        self.appli.depl_aleat_canon(self.id)
        # إخفاء القذيفة (بإرسالها خارج اللوحة) :
        self.boss.coords(self.obus, -10, -10, -10, -10)

    def effacer(self):
        "faire disparaître le canon du canevas"
        self.boss.delete(self.buse)
        self.boss.delete(self.corps)
        self.boss.delete(self.obus)

```

```
class AppBombardes(Frame):
    '''Fenêtre principale de l'application'''
    def __init__(self, larg_c, haut_c):
        Frame.__init__(self)
        self.pack()
        self.xm, self.ym = larg_c, haut_c
        self.jeu = Canvas(self, width =self.xm, height =self.ym,
                          bg ='ivory', bd =3, relief =SUNKEN)
        self.jeu.pack(padx =4, pady =4, side =TOP)

        self.guns ={}           # قاموس المدافع الموجودة
        self.pupi ={}           # قاموس الطاولات الموجودة
        self.specificites()     # كائنات مختلفة في أصناف مشتقة

    def specificites(self):
        "instanciation des canons et des pupitres de pointage"
        self.master.title('<<< Jeu des bombardes >>>')
        id_list =[("Paul","red"),("Roméo","cyan"),
                  ("Virginie","orange"),("Juliette","blue")]
        s = False
        for id, coul in id_list:
            if s:
                sens =1
            else:
                sens =-1
            x, y = self.coord_aleat(sens)
            self.guns[id] = Canon(self.jeu, id, x, y, sens, coul)
            self.pupi[id] = canon03.Pupitre(self, self.guns[id])
            s = not s           # تغيير الجانب في كل تكرار

    def depl_aleat_canon(self, id):
        "déplacer aléatoirement le canon <id>"
        gun =self.guns[id]
        dx, dy = randrange(-60, 61), randrange(-60, 61)
        # تحريك (مع تحديث الإحداثيات الجديدة) :
        x, y = gun.deplacer(dx, dy, True)
        return x, y

    def coord_aleat(self, s):
        "coordonnées aléatoires, à gauche (s =1) ou à droite (s =-1)"
        y =randrange(int(self.ym /2), self.ym -20)
        if s == -1:
            x =randrange(int(self.xm *.7), self.xm -20)
        else:
            x =randrange(20, int(self.xm *.3))
        return x, y

    def goal(self, i, j):
        "le canon n°i signale qu'il a atteint l'adversaire n°j"
        # de quel camp font-ils partie chacun ?
        ti, tj = self.guns[i].sens, self.guns[j].sens
        if ti != tj :               # إذا كانوا في اتجاهين متعاكسين :
            p = 1                   # يربح نقطة واحدة
        else:                       # إذا كانوا في نفس الاتجاه :
            p = -2                  # نضرب حليف !!
        self.pupi[i].attribuerPoint(p)
        # الذي أصيب سوف يخسر نقطة على أي حال :
        self.pupi[j].attribuerPoint(-1)

    def dictionnaireCanons(self):
        "renvoyer le dictionnaire décrivant les canons présents"
```

```
118#         return self.guns
119#
120# if __name__ =='__main__':
121#     AppBombardes(650,300).mainloop()
```

تعليقات

- السطر 6 : شكل الاستدعاء المستخدم في السطر 4 يسمح لنا بإعادة تعريف صنف جديد وهو **Canon()** المشتق من سابقه، مع الاحتفاظ بنفس الاسم. وبهذه الطريقة، ينبغي أن أجزاء التعليمات البرمجية التي تستخدم هذا الصنف لا يمكن تغييرها (لا يمكنك ذلك إذا استخدمت على سبيل المثال : « `from canon03 import *` ».)
- الأسطر من 11 إلى 16 : الأسلوب المعرف هنا الذي يحمل نفس الاسم هو أسلوب للصنف الأصل. وسيتم استبداله في صنف جديد (يمكننا القول أن الأسلوب **deplacer()** مثقل) عند تنفيذ هذا النوع من التغيير فإنه يهدف بشكل عام للتأكد من أن الأسلوب الجديد يقوم بنفس العمل كما في السابق عندما يتم استدعاؤه بنفس الطريقة الأخيرة. وهذا يضمن أن التطبيقات تستخدم الصنف الأصل. تستطيع أيضا استخدام الصنف البنت، دون تعديل نفسها.
- نحصل على هذه النتيجة عن طريق إضافة برامتر واحد أو أكثر، والقيم الافتراضية تجبر السلوك القديم. لذلك، عندما لا نقدم أي برامتر للبرامتر **rel**، البرامترات **x** و **y** يستخدمون كإحداثيات مطلقة (السلوك القديم للأسلوب). من جانب آخر، إذا كنت تقدم لـ **rel** برامتر صحيح، يتم التعامل مع البرامترات **x** و **y** كتزوح نسبي (سلوك جديد).
- الأسطر من 17 إلى 33 : سيتم إنشاء التنقلات المطلوبة بشكل عشوائي. لذلك نحن بحاجة لتوفير نظام حاجز، بحيث ينتقل الكائن ولا يخرج من اللوحة.
- السطر 42 : نحن نشير إلى الإحداثيات الجديدة الناتجة للبرنامج المستدعي ، قد يكون جيدا أن المدفع يتنقل دون معرفة موقعه الأولي.
- الأسطر 44 إلى 49 : وهذه مرة أخرى نتجاوز فيها أسلوب موجود في صنف الأصل، وذلك للحصول على سلوكيات مختلفة : بعد كل طلقة، نحن لا نقوم بتشتيت كل المدافع الحالية، لكن فقط الذي أطلق النار.
- الأسطر من 51 إلى 55 : أسلوب تم إضافته تحسبا من التطبيقات التي ترغب في تثبيت أو إزالة مدافع على مدار اللعبة .
- السطر 57 والذي يليه : هذا الصنف الجديد تم تصميمه من البداية بحيث يمكن بسهولة أن يشتق ، وهذا هو سبب في أننا قسمنا المنشئ إلى جزئين : الأسلوب **__init__()** التي تحتوي على التعليمات البرمجية المشتركة بين جميع الكائنات والتي سيتم تمثيل من هذا الصنف الذي يجب تمثيله من الصنف المشتق الممكن. الأسلوب **specificites()** يحتوي على أجزاء من الكود أكثر تحديدا : الهدف من هذا الأسلوب هو واضح وهو أن يتم تجاوز الأصناف المشتقة الممكنة .

لعبة البينغ

في الصفحات التالية، سوف تجد سكريبت لبرنامج صغير كامل. تم توفير هذا البرنامج كمثال على ما يمكن أن يتم النظر إلى تطوير نفسك كمشروع شخصي. وهذا يريك مرة أخرى كيفية يمكنك استخدام أصناف متعددة التي يمكن لمنشئ السكريبت تنظيمه. لكنه يظهر لكم كيف يمكن إنشاء تطبيق واجهة مستخدم رسومية بحيث يمكن تغيير حجم كل شيء فيها .

المبدأ

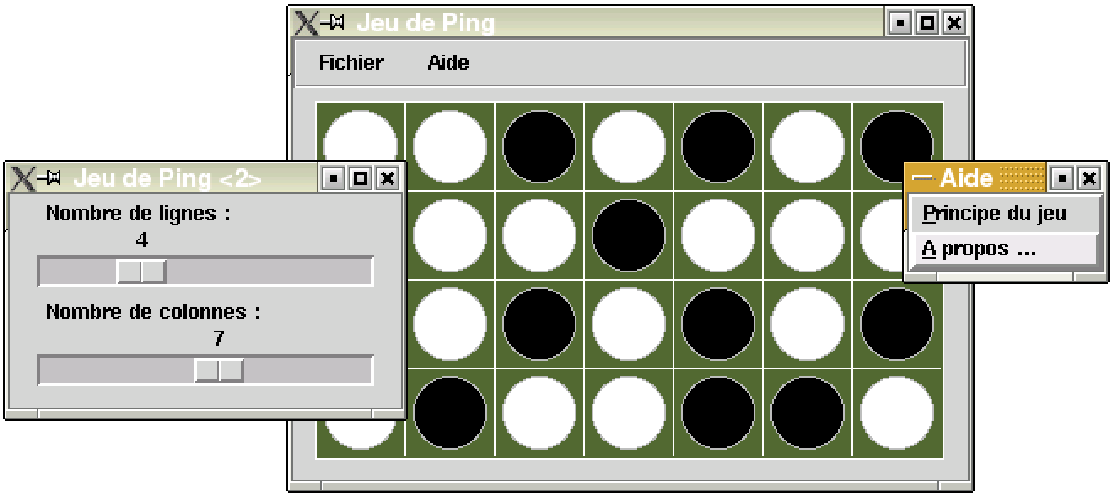
اللعبة التي تنفذ هنا هي أشبه بتمرين رياضيات. إنها تلعب على شبكة من الأحجام متغيرة، وكل خانة بها بيدق. وهذه البيادق بها وجهان الأبيض والأسود (مثل بيادق لعبة Othello/Reversi) في بداية التمرين تكون كلها بالوجه الأبيض.

عند النقر على بيدق. البيادق الأربعة المجاورة تعود. اللعبة هي إعادة جميع البيادق، بالضغط على بعض منها.

التمرين سيكون سهلا جدا من شبكة بـ 2×2 مربعات (يكفي الضغط على كل واحد من القطع الأربعة). سيكون أكثر صعوبة مع شبكة أكبر، ومستحيل مع البعض منهم. يجب أن تحدد أيها.

لا تهمل النظر على شبكة $1 \times n$.

> يمكنك العثور على المناقشة الكاملة للعبة البينغ، ونظريتها وامتداداتها. في مجلة *la science no298* لشهر أوت 2002، الصفحات من 98 إلى 102، أو على الموقع الألكتروني لجامعة *Lille* :
> <*http://www2.lifl.fr/~delahaye/dnalor/JeuAEpisodes.pdf*



برمجة

عند برمجة مشروع برمجي، حاول دائما وصف نهجك (طريقك) بأكبر قدر من الوضوح. أبدأ مع مواصفات مفصلة ولا تهمل التعليق على التعليمات البرمجية، عند البرمجة (و ليس بعد).

يمكنك أن تفرض على نفسك أن تعبر عما يريد الجهاز القيام به، والذي يساعدك على تحليل المشاكل وهيكل التعليمات البرمجية لكودك بشكل صحيح .

مواصفات البرنامج الذي تريد تطويره

- سيتم البناء على أساس نافذة رئيسية مع لوحة اللعب وشريط القوائم.
- يجب أن توسغ الإدارة من قبل المستخدم، وخانات اللوحة تبقى مربعة.
- يجب على خيارات القائمة أن تسمح بـ :
  - تحديد حجم الشبكة (عدد المربعات).
  - إعادة ضبط اللعبة (و هذا يني أن تصبح جميع البيادق بالوجه الأبيض).
  - إظهار مبدأ اللعبة في نافذة مساعدة.
  - إنهاء (إغلاق التطبيق) .
- سوف تقوم بإنشاء ثلاثة أصناف :
  - الصنف الرئيسي.
  - صنف شريط القوائم.
  - صنف للوحة اللعبة .
- ستم وضع لوحة اللعبة على لوحة، واللوحة مثبتة على إطار (frame) . نعتبر أن تغيير الحجم يتم التحكم به من خلال المستخدم، والإطار يحتل كل مرة كل المساحة المتاحة : أي ستقدم إلى المبرمج كأي مستطيل، أبعاده تسكون أساس حساب أبعاد شبكة التي سترسم.
- و بما أن هذه المربعات في الشبكة يجب أن تبقى مربعة، فمن السهل أن تبدأ بحساب حجمها الأقصى، ثم عين أبعاد اللوحة وفقا لذلك.
- إدارة نقرة الفأرة : فنقوم بربط اللوحة مع أسلوب-معالجة للخدق "ضغطة بالزر الأيسر". إحداثيات الأحداث سيتم استخدامها لتحديد أي خانة في الشبكة (رقم السطر ورقم العمود) الذي تم الضغط عليه، بغض النظر عن أبعاد الشبكة. في الخانات 8 المجاورة، البيادق سيتم "إرجاعهم" (التبديل الألوان الأسود والأبيض) .

```
#########################################
#  Jeu de ping                          #
#  Références : Voir article de la revue  #
#  <Pour la science>, Aout 2002         #
#                                       #
# (C) Gérard Swinnen (Verviers, Belgique) #
# http://www.ulg.ac.be/cifen/inforef/swi  #
#                                       #
#  Version du 29/09/2002 - Licence : GPL  #
#########################################

from tkinter import *

class MenuBar(Frame):
    """Barre de menus déroulants"""
    def __init__(self, boss =None):
        Frame.__init__(self, borderwidth =2, relief =GROOVE)
        ##### قائمة <الملف> #####
        fileMenu = Menubutton(self, text ='Fichier')
        fileMenu.pack(side =LEFT, padx =5)
        me1 = Menu(fileMenu)
        me1.add_command(label ='Options', underline =0,
                        command = boss.options)
        me1.add_command(label ='Restart', underline =0,
                        command = boss.reset)
        me1.add_command(label ='Terminer', underline =0,
                        command = boss.quit)
        fileMenu.configure(menu = me1)

        ##### قائمة <مساعدة> #####
        helpMenu = Menubutton(self, text ='Aide')
        helpMenu.pack(side =LEFT, padx =5)
        me1 = Menu(helpMenu)
        me1.add_command(label ='Principe du jeu', underline =0,
                        command = boss.principe)
        me1.add_command(label ='A propos ...', underline =0,
                        command = boss.aPropos)
        helpMenu.configure(menu = me1)

class Panneau(Frame):
    """Panneau de jeu (grille de n x m cases)"""
    def __init__(self, boss =None):
        # هذه لوحة اللعبة تتكون من إطار يمكن تغيير حجمه يحتوي على لوحة .
          عند كل تغيير في حجم الإطار, نحن نحسب أكبر حجم ممكن لمربعات الشبكة,
          و تكييف أبعاد اللوحة وفقا لذلك .
        Frame.__init__(self)
        self.nlig, self.ncol = 4, 4         # شبكة أولية 4 = x 4
        # ربط الحدث <resize> لمعالج مناسب :
        self.bind("<Configure>", self.redim)
        # اللوحات :
        self.can =Canvas(self, bg ="dark olive green", borderwidth =0,
                         highlightthickness =1, highlightbackground ="white")
        # ربط الحدث <clic de souris> لمعالجة :
        self.can.bind("<Button-1>", self.clic)
        self.can.pack()
        self.initJeu()
```

```
    def initJeu(self):
        "Initialisation de la liste mémorisant l'état du jeu"
        self.etat =[]                  # صنع قائمة من قوائم
        for i in range(12):             # (يعادل جدول من
            self.etat.append([0]*12)    #  12 خط x 12 عمود)

    def redim(self, event):
        "Opérations effectuées à chaque redimensionnement"
        # الخصائص المرتبطة مع حدث إعادة التكوين يحتوي على أبعاد الجديدة
للإطار :
        self.width, self.height = event.width -4, event.height -4
        # ويستخدم فرق 4 بيكسلات للتعويض عن سمك "الحدود" المحيط باللوحة .
        self.traceGrille()

    def traceGrille(self):
        "Dessin de la grille, en fonction des options & dimensions"
        # العرض والإرتفاع الأقصى للمربعات :
        lmax = self.width/self.ncol
        hmax = self.height/self.nlig
        # جانب لمربع يساوي أصغر هذه الأبعاد :
        self.cote = min(lmax, hmax)
        # -> إنشاء أبعاد جديدة للوحة :
        larg, haut = self.cote*self.ncol, self.cote*self.nlig
        self.can.configure(width =larg, height =haut)
        # تخطيط الشبكة :
        self.can.delete(ALL)                # محو الرسوم السابقة
        s =self.cote
        for l in range(self.nlig -1):       # خطوط أفقية
            self.can.create_line(0, s, larg, s, fill="white")
            s +=self.cote
        s =self.cote
        for c in range(self.ncol -1):       # خطوط عمودية
            self.can.create_line(s, 0, s, haut, fill ="white")
            s +=self.cote
        # تتبع جميع القطع, بيضاء أو سوداء حسب حالة اللعبة :
        for l in range(self.nlig):
            for c in range(self.ncol):
                x1 = c *self.cote +5            # حجم القطع(البيدق) =
                x2 = (c +1)*self.cote -5        # حجم المربع -10
                y1 = l *self.cote +5            #
                y2 = (l +1)*self.cote -5
                coul =["white","black"][self.etat[l][c]]
                self.can.create_oval(x1, y1, x2, y2, outline ="grey",
                                     width =1, fill =coul)

    def clic(self, event):
        "Gestion du clic de souris : retournement des pions"
        # نبدأ بتحديد السطر والعمود :
        lig, col = int(event.y/self.cote), int(event.x/self.cote)
        # بعد ذلك نقوم بمعالجة 8 المربعات المجاورة :
        for l in range(lig -1, lig+2):
            if l <0 or l >= self.nlig:
                continue
            for c in range(col -1, col +2):
                if c <0 or c >= self.ncol:
                    continue
```

```
                if l ==lig and c ==col:
                    continue
                # عكس البيدق بعكس منطقي :
                self.etat[l][c] = not (self.etat[l][c])
        self.traceGrille()


class Ping(Frame):
    """corps principal du programme"""
    def __init__(self):
        Frame.__init__(self)
        self.master.geometry("400x300")
        self.master.title(" Jeu de Ping")

        self.mbar = MenuBar(self)
        self.mbar.pack(side =TOP, expand =NO, fill =X)

        self.jeu =Panneau(self)
        self.jeu.pack(expand =YES, fill=BOTH, padx =8, pady =8)

        self.pack()

    def options(self):
        "Choix du nombre de lignes et de colonnes pour la grille"
        opt =Toplevel(self)
        curL =Scale(opt, length =200, label ="Nombre de lignes :",
              orient =HORIZONTAL,
              from_ =1, to =12, command =self.majLignes)
        curL.set(self.jeu.nlig)     # الموقع الأولي للمؤشر
        curL.pack()
        curH =Scale(opt, length =200, label ="Nombre de colonnes :",
              orient =HORIZONTAL,
              from_ =1, to =12, command =self.majColonnes)
        curH.set(self.jeu.ncol)
        curH.pack()

    def majColonnes(self, n):
        self.jeu.ncol = int(n)
        self.jeu.traceGrille()

    def majLignes(self, n):
        self.jeu.nlig = int(n)
        self.jeu.traceGrille()

    def reset(self):
        self.jeu.initJeu()
        self.jeu.traceGrille()

    def principe(self):
        "Fenêtre-message contenant la description sommaire du principe du jeu"
        msg =Toplevel(self)
        Message(msg, bg ="navy", fg ="ivory", width =400,
            font ="Helvetica 10 bold",
            text ="Les pions de ce jeu possèdent chacun une face blanche et "\
            "une face noire. Lorsque l'on clique sur un pion, les 8 "\
            "pions adjacents se retournent.\nLe jeu consiste a essayer "\
            "de les retouner tous.\n\nSi l'exercice se révèle très facile "\
            "avec une grille de 2 x 2 cases. Il devient plus difficile avec "\
            "des grilles plus grandes. Il est même tout à fait impossible "\
```

```
            "avec certaines grilles.\nA vous de déterminer lesquelles !\n\n"\
            "Réf : revue 'Pour la Science' - Aout 2002")\
            .pack(padx =10, pady =10)

    def aPropos(self):
        "Fenêtre-message indiquant l'auteur et le type de licence"
        msg =Toplevel(self)
        Message(msg, width =200, aspect =100, justify =CENTER,
            text ="Jeu de Ping\n\n(C) Gérard Swinnen, Aout 2002.\n"\
            "Licence = GPL").pack(padx =10, pady =10)

if __name__ == '__main__':
    Ping().mainloop()
```

> تذكير
>
> إذا كنت ترغب في تجربة هذه البرامج بدون إعادة كتابتها. يمكنك العثور على كودها على : *http://www.inforef.be/swi/python.htm*.

16

إدارة قواعد البيانات

قواعد البيانات هي أدوات تستخدم بشكل متزايد. يتم استخدامها لتخزين البيانات العديدة في حزمة واحدة منظمة بشكل جيد. عندما يتعلق الأمر بقواعد البيانات العلائقية، يمكن تماما تجنب "جحيم التكرار". ربما قد واجهة بالفعل هذه المشكلة : تم تخزين نفس البيانات في ملفات مختلفة. وعندما تريد تعديل أو حذف أي من هذه البيانات، يجب عليك فتحها وتعديل أو حذفها في جمبع الملفات التي تحتويها ! وإحتمال الخطأ كبير جدا، وهذا الأمر يؤدي إلى التضارب، ناهيك عم ضياع الوقت. والحل لهذه المشكلة هي قواعد البيانات. بيثون يتيح لك طرق مختلفة لاستخدام موارد الكثير من الأنظمة، لكننا لن نختبر سوى مثالين : *SQLite* و *PostgreSQL*.

قواعد البيانات

هنالك العديد من قواعد البيانات. يمكننا على سبيل المثال اعتبار قاعدة البيانات ملفا يحتوي على قائمة من الأسماء والعناوين.

إذا كانت القائمة ليست طويلة جدا، وإذا كنت لا تريد إمكانية تنفيذ عمليات البحث على أساس معايير معقدة، فمن نافلة القول انه يمكن الوصول إلى هذا النوع من البيانات باستخدام تعليمات بسيطة مثل تلك التي ناقشناها في الصفحة 114.

و سيتعقد الوضع بسرعة كبيرة إذا كنا نريد تحديد وفرز البيانات، خاصة إذا ازداد عددها. وستزداد الصعوبة إذا تم سرد البيانات في مجموعات مختلفة متصلة بواسط عدد من العلاقات، وإذا كان هنالك العديد من المستخدمين بحاجة إلى الوصول إليها في نفس الوقت.

على سبيل المثال، تخيل أن مدير مدرستك يتعهد لكم بتطوير نظام نشرة محوسبة. حان وقت التفكير قليلا. كنت قد أدركت بسرعة أنه يجب تنفيذ مجموعة من الجداول المختلفة : جدول أسماء الطلاب (و التي قد تحتوي بطبيعة الحال معلومات خاصة بهؤلاء الطلاب : العنوان، تاريخ الميلاد، إلخ ...) وجدول يحتوي على قائمة الدروس (مع اسم الأستاذ، وعدد ساعات التعليم في

الأسبوع الخ .). وجدول لتخزين الأعمال (مع أهميتها، وتاريخها، ومحتواها إلخ .). وجدول يصف كيف يمكن جمع الطلاب في مجموعات حسب الفصول أو الخيارات، والدروس التي أخذها كل طالب، إلخ.

يجب أن تعرف أن هذه الجداول ليسا مستقلة. بل ترتبط بالعمل الذي قام به الطالب مع دروسه المختلفة. لتحديد مقدار نشرة الطالب، إذًا لابد من استخراجها من جداول العمل، بالطبع، لكن مع المعلومات الموجود في الجداول الأخرى (هذه الدورات والصفوف والخيارات وإلخ .).

سوف نرى لاحقا كيفية تمثيل جداول والعلاقات بينها .

SGBDR – RGB - نموذج عميل\خادم (سيرفر)

البرامج الحاسوبية قادرة على إدارة مجموعات من البيانات المعقدة (معقدة كثيرا أيضا) ، وندعو هذه البرامج بـ SGBDR (و هي اختصار لكلمة فرنسية معناها إدارة أنظمة قواعد البيانات العلائقية). وهذه التطبيقات المعلوماتية مهمة جدا للشركات. بعض من هذه الشركات المتخصصة في صناعتها : IBM وأوركل ومايكروسوفت و informix و Sybase ...) وعادة ما تباع بأسعار مرتفعة. وقد تم تطوير البرامج الأخرى في مراكز أبحاث وتدريس جامعي (MySQL، SQLite، PostgreSQL...). وعادة ما تكون مجانية.

هذه الأنظمة لدى كل واحدة منها خصائصها وأدائها، ولكن معظمها تعمل على نموذج عميل/سيرفر : وهذا يعني الجزء الأكبر من البرنامج (يتم إدارته من خلال قواعد البيانات) يتم تثبيته في مكان واحد، من حيث المبدأ على آله قوية (و هذا يشكل الخادم(السيرفر) بأكمله)، في حين أن الأخر أكثر بساطة من ذلك بكثير، فهو يتم تثبيته على أي عدد من محطات العمل ، تدعى العملاء (clients).

و يتم ربط العملاء بالخادم (سيرفر)، بشكل دائم أو لا، عن بطرق مختلفة وبروتوكولات (ربما عن طريق الإنترنت). كل واحد منهم يمكنه الوصول إلى جزء مهم جدا أو أقل أهمية، مع موافقة أو لا لتعديل بعضها، إضافة أو حذف، اعتمادا على قواعد محددة جدا، تم تعريفها عن طريق مدير قاعدة البيانات.

الخادم والعملاء هي في الواقع تطبيقات منفصلة التي تتبادل المعلومات. تخيل على سبيل المثال أنك أحد مستخدمي النظام.

للوصول إلى البيانات، يجب تشغيل تطبيق عميل في أي محطة عمل. عند تشغيله، يبدأ تطبيق العميل عن طريق تأسيس اتصال مع الخادم وقاعدة البيانات[82]. عندما يتم تأسيس الاتصال، يمكن للتطبيق العميل الاستعلام عن الخادم عن طريق إرسال طلب في شكل متفق عليه. فعلى سبيل المثال، عند البحث عن معلومة دقيقة. يتم تنفيذ الخادم عن طريق البحث في البيانات المناظرة في قواعد البيانات، ثم يرجع الإجابة للعميل.

---

82 قد تحتاج إلى إدخال بعض المعلومات للوصول : عنوان سيرفر على الشبكة، اسم قاعدة البيانات، اسم المستخدم، كلمة المرور ...

و يعتبر هذا ردا على المعلومات المطلوبة، أو رسالة خطأ في حالة الفشل.

الاتصالات بين العميل والخادم هي عبارة على طلبات وردود. الطلبات هي تعليمات حقيقية يتم إرسالها من العميل إلى الخادم، وليس فقط لاستخراج من قواعد البيانات، لكن أيضا إضافة أو حذف أو تعديل وإلخ .

لغة SQL

بعد قراءة ما سبق، سوف تفهم أننا لن نشرح في هذه الصفحات كيفية صنع برنامج خادم. وهذا من عمل المتخصصين (على سبيل المثال،كأنك ستطور لغة برمجة جديدة). وأما تطوير برنامج عميل، هو شيء في متناول اليد، ويمكنك تحقيق فائدة كبيرة. ويجب أن تعرف أن معظم التطبيقات "الجدية" تعمل على قواعد بيانات بمختلف التعقيد : حتى الألعاب يجب عليها تخزين الكثير من البيانات والحفاظ على العلاقات بينها.

اعتمادا على احتياجات تطبيقك، سيكون لديك الاختيار، إما أن تتصل بخادم بعيد يتمكن الاتصال به العديد من المستخدمين، وإما أن تصنع خادما محليا أقل أو أكثر كفاءة. في حالة تطبيق منفرد، يمكنك استخدام برنامج خادم مثبت على نفس الجهاز الذي يوجد به تطبيقك، أو ببساطة أكثر، استخدام مكتبة خادم متوافقة مع لغة البرمجة الخاصة بك. سوف ترى في جميع الحالات، أن أليات التنفيذ ستبقى في الأساس نفسها.

يمكن للمرء أن يقلق، في الواقع ، إنه بالنظر إلى التنوع الكبير من الخوادم الموجودة، فمن الضروري استخدام لغات مختلفة وبروتوكولات لإرسال الطلبات إلى كل واحد منهم. لكن لحسن الحظ، بذلت جهود كبيرة لتوحيد تطوير لغة الاستعلام المشترط، والتي ما تسمى SQL (Structured Query Language - لغة الاستعلام الهيكلية)[83]. فيما يتعلق ببيثون، تم تقديم جهود إضافية لتوحيد الإجراءات للوصول إلى الملقمات نفسها، وعلى واجهة مشتركة (DBAPI[84]).

لذا يجب عليك حفظ بعض أساسيات اللغة للاستمرار، ولكن لا ينبغي أن تقلق. بالتأكيد سوف تجد فرصة للالتقاء بـ SQL في مجالات أخرى (على سبيل المثال، المكتبية). في إطار محدود من هذه الدورة، يجب عليك مراجعة بعض تعليمات لغة SQL بسيطة لفهم الأليات الأساسية وربما جعل بعض المشاريع مثيرة للاهتمام .

83 توجد بعض الإختلافات بين التطبيقات المختلفة من SQL، للاستعمالات المحددة جدا، لكن القاعدة (الأساس) يبقى نفسه .

84 بيثون DataBase Application Programming Interface Specification يعرف مجموعة من القواعد السلوكية لمطوري الوحدات للوصول إلى SGBDR المختلفة، حيث أن هذه الوحدات قابلة للتبادل . و بالتالي، نفس تطبيق بيثون سوف يكون قادرا على استخدام SGBDR أو آخر، بسعر تبادل بسيط من الوحدات .

SQLite

مكتبة بيثون القياسية تشمل محرك قاعدة بيانات علائقية يدعى SQLite[85]، تم تطويرها بشكل مستقل بلغة سي، وتنفذ العديد من معايير SQL-92.

و هذا يعني أنه يمكنك كتابة تطبيق ببيثون يحتوي على SGBDR مدمج، دون الحاجة إلى تثبيت أي شيء آخر، وهذا سيحسن الأداء.

سوف ترى في نهاية الفصل كيف تسير الأمور إذا كان التطبيق الخاص بك يجب أن يستخدم بدلا من ذلك خادم قواعد البيانات التي تم استضافتها في جهاز آخر، ولكن المبادئ تبقى نفسها. كل هذا سوف تتعلمه مع SQLite وسيكون قابل للنقل دون تعديل، إذا كنت تريد لاحقا العمل مع SGDBR أكثر "فرض" مثل PostgreSQL أو MySQL أو Oracle.

لنبدأ على الفور لاستكشاف أساسيات هذا النظام، على سطر الأوامر. سوف نكتب فيما بعد سكريبت صغير لإدارة قاعدة بيانات بسيطة مع جدولين .

إنشاء قاعدة بيانات - كائنات "اتصال" و"مؤشر"

و كما كنت تتوقع، يجب استدعاء وحدة للوصول إلى مميزاتها :

```
>>> import sqlite3
```

> الرقم في نهاية الاسم هو رقم الإصدار الحالي من وحدة واجهة في وقت كتابة هذه الأسطر. فمن الممكن أن يتم تغيير هذا في الإصدارات المستقبلية من بيثون .

ثم يجب عليك أن تقرر اسم الملف الذي تريد تعيينه إلى قاعدة البيانات. SQLite يقوم بحفظ جميع جداول قاعدة البيانات في ملف واحد متعدد المنصات الذي يمكنك أن تقوم بحفظ أي شيء تريد (وهذا يجب أن يبسط إلى حد كبير حياتك للأرشيفات !) :

```
>>> fichierDonnees ="E:/python3/essais/bd_test.sq3"
```

اسم الملف يمكن أن يتضمن اسم المسار وأي امتداد. ومن الممكن استخدام اسم خاص `:memory:`، يشير إلى أن يتم معالجة قواعد البيانات في الذاكرة العشوائية (رام) فقط. وبذلك يمكنك اختصار الوقت والوصول إلى البيانات، والتطبيق سيكون سريعا جدا، وقد يكون ذا فائدة في سياق برنامج لعبة على سبيل المثال، شرط أن تكون هنالك آلية خاصة للحفظ على القرص.

[85] SQLite (http://www.sqlite.org/) هو محرك قواعد بيانات الأكثر استخداما في العالم . يتم استخدامه في العديد من الأدوات مثل Firefox, Skype, Google Gears، و في بعض منتجات أبل و أدوبي و مكافي و في المكتبات القياسية في العديد من الغات البرمجة مثل PHP و بيثون . و هو أيضا الأكثر شعبية في النظم المضمنة، بما في ذلك الهواتف الذكية الحديثة . و هو مجاني و خالي من الحقوق .

سوف تقوم إذا بصنع كائن-اتصال، بمساعد دالة-صنع **()connect**. هذا الكائن يتفاعل بين البرنامج وقاعدة البيانات . العملية مماثلة تماما لفتح ملف نصي، ومثيل كائن سيصنع ملف التخزين (إذا كان الملف غير موجود) :

```
>>> conn =sqlite3.connect(fichierDonnees)
```

تم الآن وضع كائن الاتصال في مكانه، وسوف تكون قادرا على التفاعل معه باستخدام SQL. سيكون هذا ممكنا مباشرة عن طريق استخدام بعض أساليب هذا الكائن[86]، لكن من المفضل أن تضع في مكانه لتحاور مع كائن-واجهة أخرى يسمى المؤشر. بل هو نوع من الذاكرة العازلة، لتخزين البيانات في الذاكرة بشكل مؤقت عند القيام بمعالجتها، فضلا عن عمليات تقوم لها عليها، قبل نقلها إلى قاعدة البيانات النهائية. هذه التقنية يجل من الممكن إلغاء إذا لزم الأمر عملية أو أكثر التي مهي غير كافية، وإعادتها إلى معالجتها، دون أن تتأثر قاعدة البيانات (يمكنك معرفة المزيد عم هذا المفهوم من خلال إحدى وثائق التي تتعامل مع لغة SQL) .

```
>>> cur =conn.cursor()
```

قاعدة البيانات تتكون دائما من جدول أو أكثر، يحتوي على السجلات (أو المحفوظات)، وهي تحتوي على أنواع مختلفة من المجالات. وهذه المفاهيم ربما كنت على دراية بها إذا كنت قد عملت مع أي جدول : السجلات يتم حفظها في أسطر الجدول، والمجالات في خلايا السطر. سوف نكتب أول استعلام SQL لنطلب منه إنشاء جدول جديد :

```
>>> cur.execute("CREATE TABLE membres (age INTEGER, nom TEXT, taille REAL)")
```

يتم التعبير عن الاستعلام في سلسلة نصية كلاسيكية، والتي نريد تمريرها للمؤشر عبر أسلوبه **()execute**. لاحظ جيدا أن SQL يتجاهل حالة الأحرف، بحيث يمكن ترميز استعلامات SQL بحروف كبيرة أو صغيرة (أو معا). اخترنا شخصيا الكتابة بحروف كبيرة تعليمات هذه اللغة، وذلك للتفرقة بين تعليمات بيثون المحيطة بها، ولكن بالطبع يمكنك أن تتبع عادات أخرى.

كما يرجى ملاحظة أن أنواع البيانات لا تحمل نفس الأسماء في بيثون وفي SQL. لا ينبغي على الترجمة أن تزعجك كثيرا. ملاحظة بسيطة وهي أن السلاسل النصية يتم ترميزها إفتراضيا بـ Utf-8، حسب الاتفاقية ذاتها مع الملفات النصية (انظر للصفحة 122).

يمكننا الآن إدخال السجلات :

```
>>> cur.execute("INSERT INTO membres(age,nom,taille) VALUES(21,'Dupont',1.83)")
>>> cur.execute("INSERT INTO membres(age,nom,taille) VALUES(15,'Blumâr',1.57)")
>>> cur.execute("INSERT Into membres(age,nom,taille) VALUES(18,'Özémir',1.69)")
```

[86] وحدة SQLite توفر بعض الأساليب المختصرة للوصول إلى البيانات دون استخدام المؤشر (أو على نحو أدق, وذلك باستخدام مؤشر ضمني) . هذه الأساليب لا تتوافق مع التقنيات القياسية, و نحن نفضل تجاهل ذلك هنا .

انتبه، في هذه المرحلة من العمليات، سيتم حفظ السجلات في مؤشر عازل، لكنا لم تنقل بعد إلى قاعدة البيانات. لذا يمكنك إلغائها تماما، إذا لزم الأمر، كما سنرى بعد قليل. وسيتم تشغيل نقل البيانات من خلال الأسلوب **commit()** لكائن الاتصال ::

```
>>> conn.commit()
```

و بعد الانتهاء من العمل يمكنك إغلاق المؤشر، وكذلك الاتصال.[87]

```
>>> cur.close()
>>> conn.close()
```

الاتصال بقاعدة بيانات موجودة

بعد العمليات أعلاه، تم إنشاء ملف يسمى **bd_test.sq3** في موقع محدد في جهازك. لقد قمت الآن بالخروج من بيثون وربما قد أغلقت حاسوبك : البيانات التي تم حفظها، كيف يمكننا الوصول إليها مرة أخرى ؟ الأمر في غاية البساطة : يكفي أن تستخدم بالضبط هذه التعليمات :

```
>>> import sqlite3
>>> conn =sqlite3.connect("E:/python3/essais/bd_test.sq3")
>>> cur =conn.cursor()
```

يتم تنفيذ الاستعلام بالطبع بمساعدة استعلامات SQL، الذي يعكي للأسلوب **execute()** للمؤشر، دائما في شكل سلسلة نصية :

```
>>> cur.execute("SELECT * FROM membres")
```

هذا الاستعلام يقوم بطلب تحديد مجموعة معينة من السجلات، سيتم تحويلها من قاعدة البيانات إلى المؤشر. في هذه الحالة، التحديد ليس عنصر واحد، لأننا طلبنا أن يتم استرداد جميع سجلات الجدول membres .

> تذكر أن الرمز * يستخدم كثيرًا في المعلوماتية ك"جوكر" بمعني "كل" .

السجلات المحددة هي الآن في المؤشر. فإذا أردنا أن نراها، يجب علينا استخراجها. وهذا يتم بطريقتين، وقد تبدو للوهلة الأولى مختلفة، لكن في الواقع أن الطريقتين لكائن-المؤشر يتم صنعها من بيثون هي مكررة، وهذا يعني، جهاز توليد المتسلسلات.[88]

---

87 التطبيقات التي تستخدم قواعد بيانات كبيرة غالبا ما تكون تطبيقات متعددة المستخدمين . وسوف نرى لاحقا (صفحة Error: Reference source not found) أن مثل هذه التطبيقات تنفذ عدة "أبناء" لتنفيذ متزامن للبرنامج، وتدعى المواضيع، من أجل التعامل مع الطلبات الموازية من عدة مستخدمين مختلفين . وبالتالي سوف يكون لكل واحد كائنات اتصالات و مؤشر داخل البرنامج نفسه، و أنه لن يكون هنالك تضارب . في حالة SQLite، و هو نظام مستخدم منفرد، إغلاق الاتصالات يتسبب أيضا بإغلاق الملف الذي يحتوي على قاعدة البيانات، و التي سوف يختلف عن النظام الكبير .

88 التكرارات هي جزء من مميزات المتقدمة لبيثون . نحن لن ندرسها في هذا الكتاب، و كذلك العديد من الأدوات الأخرى المثيرة للاهتمام، مثل تعريف الوظيفي للقوائم، و الديكورات، إلخ . و سوف تظل أشياء كثيرة لا تزال لإستكشافها' إذا كنت تريد استكشاف

يمكنك الذهاب مباشرة إلى التسلسل المنتج، بمساعدة حلقة **for** الكلاسيكية، وسوف تحصل على مجموعة من المصفوفات المغلقة :

```
>>> for l in cur:
...     print(l)
...
(21, 'Dupont', 1.83)
(15, 'Blumâr', 1.57)
(18, 'Özémir', 1.69)
```

... أو يتم جمعها في قائمة أو مصفوفة مغلقة لمزيد من المعالجة (بمساعدة الدالات المدمجة **list()** و **tuple()**) :

```
>>> cur.execute("SELECT * FROM membres")
>>> list(cur)
[(21, 'Dupont', 1.83), (15, 'Blumâr', 1.57), (18, 'Özémir', 1.69)]
```

بطريقة أكثر كلاسيكية، يمكنك أيضا استدعاء الدالة **fetchall()** للمؤشر، التي تقوم بإرجاع قائمة مصفوفات مغلقة :

```
>>> cur.execute("SELECT * FROM membres")
>>> cur.fetchall()
[(21, 'Dupont', 1.83), (15, 'Blumâr', 1.57), (18, 'Özémir', 1.69)]
```

كما أن المؤشر لا يزال مفتوحا، يمكنك بالطبع إضافة سجلات إضافية :

```
>>> cur.execute("INSERT INTO membres(age,nom,taille) VALUES(19,'Ricard',1.75)")
```

في برنامج عملي، البيانات التي تريد تسجيلها يتم حفظها في متغيرات تنشأ في الغالب في متغيرات بيثون. وسوف تحتاج أيضا إلى إنشاء سلسلة نصية تحتوي على طلب استعلام SQL، لتشمل قيما من هذه المتغيرات. فمن المستحسن استخدامها لهذا الغرض في التقنيات الهادية لتنسيق السلاسل، لأن هذه قد تفتح ثغرة أمنية في برامجها، وتسمح لاقتحامها من خلال طريقة تدعى SQL Injection (حقن SQL)[89]. ولذلك يجب التأكد من تنسق استعلاماتك للوحدة الواجهة نفسها. والتقنية السليمة أدناه : سلسلة "رئيس" تستخدم علامة استفهام كعلامات التحويل، وتنسيق نفسه معتمد من قبل الأسلوب **execute()** للمؤشر:

```
>>> data =[(17,"Durand",1.74),(22,"Berger",1.71),(20,"Weber",1.65)]
>>> for tu in data:
...     cur.execute("INSERT INTO membres(age,nom,taille) VALUES(?,?,?)", tu)
...
>>> conn.commit()
```

هذه اللغة ؛

89 هذه المشكلة الأمنية تنشأ عن طريق تطبيقات الويب. الهجوم يتم باستخدام حقل نموذج HTML و يؤدي إلى حقن التعليمات SQL بالبيانات الخبيثة حيث يتوقع البرنامج أن السلاسل غير مؤذية . و مع ذلك. فمن المستحسن استخدام تقنيات برمجة أكثر أمنا. حتى ولو تطبيق بسيط لشخص واحد .

في هذا المثال، سلسلة الاستعلام تحمل 3 علامات استفهام، والتي هي علاماتنا. سوف يتم استبدالهم بـ 3 عناصر من نوع مصفوفات مغلقة في كل تكرار للحلقة، وحدة الواجهة مع SQLite يتم تحميلها مع كل متغير وفقا لنوعه.

في هذه المرحلة من العمليات، قد تعتقد أن كل ما رأيناه هو معقد للغاية لكتابة وقراءة المعلومات في ملف. لن يكون أكثر بساطة إذا استخدمت معالجات ملفات التي نعرفها ؟ نعم ولا. هذا صحيح بالنسبة للكميات الصغيرة من المعلومات التي لا تحتاج إلى تغيير كبير مع مرور الوقت. لكننا لا يمكننا الدفاع عنه إذا أخذنا مشكلة بسيطة للتعديل أو حذف أو إضافة أي سجل. في قاعدة البيانات، هذا بسيط جدا :

```
>>> cur.execute("UPDATE membres SET nom ='Gerart' WHERE nom='Ricard'")
```

لحذف سجل أو أكثر، استخدم استعلام مثل هذه :

```
>>> cur.execute("DELETE FROM membres WHERE nom='Gerart'")
```

مع ما نعرفه من الملفات النصية، يجب علينا بالتأكيد كتابة العديد من الأسطر(كود) للحصول على نفس الشيء ! ولكن هنالك الكثير مثير للاهتمام .

> انتبه
>
> لا تنسَ أن تغييرات المؤشر تحدث في الرام. وبالتالي لن يتم حفظ أي شيء بشكل دائما ما لم تقم بتشغيل التعليمة ***conn.commit()***. يمكنك إلغاء جميع التغييرات منذ ***commit()*** السابق. وإغلاق الإتصال باستخدام الأمر ***conn.close()***

البحث التحديدي في قاعدة بيانات

تمرين

1.16 قبل المضي قدما، وبوصفها تمرين تجميعي، سوف أطلب منك إنشاء قاعدة بيانات "Musique" تحتوي على الجدولين التاليين (هذه بعض الأعمال، لكن يجب أن تكون قادرا على التعامل مع عدد من البيانات بشكل صحيح لاختبار دالات البحث والفرز المعتمد من قبل SGBDR) :

Compositeurs
(comp (chaîne
(a_naiss (entier
(a_mort (entier

Oeuvres
(comp (chaîne
(titre (chaîne
(duree (entier
(interpr (chaîne

ابدأ بملء جدول **Compositeurs** مع البيانات التالية (اغتنم هذه الفرصة لإظهار مهاراتك التي تعلمتها من خلال كتابة سكريبت صغير لتسهيل إدخال المعلومات: تحتاج إلى حلقة!):

```
comp          a_naiss      a_mort

Mozart        1756         1791
Beethoven     1770         1827
Haendel       1685         1759
Schubert      1797         1828
Vivaldi       1678         1741
Monteverdi    1567         1643
Chopin        1810         1849
Bach          1685         1750
Shostakovich  1906         1975
```

في جدول **oeuvres**، أدخل البيانات التالية :

```
comp         titre                        duree    interpr

Vivaldi      Les quatre saisons            20      T. Pinnock
Mozart       Concerto piano N°12           25      M. Perahia
Brahms       Concerto violon N°2           40      A. Grumiaux
Beethoven    Sonate "au clair de lune"     14      W. Kempf
Beethoven    Sonate "pathétique"           17      W. Kempf
Schubert     Quintette "la truite"         39      SE of London
Haydn        La création                  109      H. Von Karajan
Chopin       Concerto piano N°1            42      M.J. Pires
Bach         Toccata & fugue                9      P. Burmester
Beethoven    Concerto piano N°4            33      M. Pollini
Mozart       Symphonie N°40                29      F. Bruggen
Mozart       Concerto piano N°22           35      S. Richter
Beethoven    Concerto piano N°3            37      S. Richter
```

يحتوي الحقلان **a_naiss** و **a_mort** على سنة الميلاد وسنة موت الملحنين. مدة تنفيذ هذا في دقائق. بالطبع يمكنك إضافة العديد من السجلات للملحنين والمؤلفين التي تردها، لكن تلك المذكورة أعلاه ينبغي أن تكون كافية لبقية المظهر.

في ما يلي، نحن نفترض أنك قد قمت بترميز البيانات في الجدولين أعلاه. إذا كانت لديك صعوبة في كتابة السكريبت المطلوب، يرجى الرجوع إلى تمرين 16.1 في Error: Reference source not found.

السكريبت الصغير بالأسفل يوفر أغراض المعلومات فقط. بل هو عميل SQL بدائي، الذي يسمح لك بالإتصال بقاعدة البيانات " musique - موسيقى" التي يجب أن تكون الآن موجودة في الدليل الخاص بك، ويتم فتح المؤشر لاستخدامه للاستعلام. لاحظ مرة أخرى أننا n'a pas été **commit()** que rien n'est transcrit sur le disque tant que la méthode invoquée.

```
# استخدام قاعدة بيانات صغيرة تقبل تعليمات SQL

import sqlite3

baseDonn = sqlite3.connect("musique.sq3")
```

```
cur = baseDonn.cursor()
while 1:
    print("Veuillez entrer votre requête SQL (ou <Enter> pour terminer) :")
    requete = input()
    if requete =="":
        break
    try:
        cur.execute(requete)          # تشغيل استعلام SQL
    except:
        print('*** Requête SQL incorrecte ***')
    else:
        for enreg in cur:             # إظهار الناتج
            print(enreg)
    print()

choix = input("Confirmez-vous l'enregistrement de l'état actuel (o/n) ? ")
if choix[0] == "o" or choix[0] == "O":
    baseDonn.commit()
else:
    baseDonn.close()
```

هذا التطبيق البسيط جدا من الواضح أنه مثال. ينبغي أن نضيف خيار لاختيار قاعدة البيانات والدليل. استخدمنا لمنع السكربت من "زرع" عندما يقوم المستخدم بترميز استعلام غير صحيح، استخدمنا هنا معالجة الاستثناءات التي قمنا بشرحها في الصفحة 125.

استعلام التحديد (select)

واحدة من أقوى تعليمات لغة SQL هي التعليمة **select**، التي سنرى الآن بعض مميزاته. وتذكر مرة أخرى أننا سنتناول هنا جزءاً صغيرًا جدا من هذا الموضوع : الوصف التفصيلي لـ SQL يجب أن يشرح في كتب عديدة.

شغل إذا السكربت أعلاه، وحلل بدقة ما يحدث عند تقديم الاستعلامات التالية :

```
select * from oeuvres

select * from oeuvres where comp = 'Mozart'

select comp, titre, duree  from oeuvres order by comp

select titre, comp from oeuvres where comp='Beethoven' or comp='Mozart'
   order by comp

select count(*) from oeuvres

select sum(duree) from oeuvres

select avg(duree) from oeuvres

select sum(duree) from oeuvres where comp='Beethoven'

select * from oeuvres where duree >35 order by duree desc

select * from compositeurs where a_mort <1800
```

```
select * from compositeurs where a_mort <1800 limit 3
```

لكل واحدة من هذه الاستعلام، للتعبير عن أفضل ما سيحدث. في الأساس، قمت بتفعيل المرشحات على قواعد البيانات للتحديد والفرز.

الاستعلامات التالية هي أكثر تطويرا، لأنها تصل جدولين في المرة الواحدة.

```
select o.titre, c.comp, c.a_naiss from oeuvres as o, compositeurs as c where o.comp
=c.comp

select comp, titre, a_naiss from oeuvres join compositeurs using(comp)

select * from oeuvres join compositeurs using(comp) order by a_mort

select comp from oeuvres intersect select comp from compositeurs

select comp from oeuvres except select comp from compositeurs

select comp from compositeurs except select comp from oeuvres

select distinct comp from oeuvres union select comp from compositeurs
```

لا يمكننا أن نطور لغة استعلام في السياق المحدود من هذا الكتاب. ومع ذلك سوف نختبر مثالا آخر لتجسيد بيثون عند استخدام قواعد البيانات، لكن على افتراض أن الوقت قد حان لإجراء اتصال بنظام خادم مستقل (و التي يمكن أن تكون على سبيل المثال خادم قواعد بيانات كبير للشركات، خادم وثائق في مدرسة، إلخ). كما أن هنالك العديد من البرامج الممتازة الحرة والمفتوحة المصدر، يمكنك البدء بسهولة استخدام خادم فاعل للغاية مثل PostgreSQL[90]. سوف يكون التمرين للاهتمام بوجه خاص إذا كنت تريد أن تأخذ عناء تثبيت برنامج خادم على جهاز منفصل عن محطة العمل الخاصة بك، وسوف تربط الإثنين باتصال عبر الشبكة من نوع TCP/IP .

مشروع برنامج عميل لـ PostgreSQL

لإنهاء هذا الفصل، فسوف نقترح عليك في الصفحات القادم مثال تطبيق عملي. لن نصنع برنامج حقيقي (الموضوع يتطلب كتاب مخصص). لكن مشروع (نموذج) تحليلي، مصمم ليبين لك كيف يمكنك "التفكير مثل مبرمج" عندما نحصل على مشكلة معقدة.

90 إن PostgreSQL هو SGBDR حرة، متاحة تحت رخصة من نوع BSD .

هذا النظام متطور جدا قادر على منافسة غيره من نظم إدارة قواعد البيانات الحرة (مثل MySQL و Firebird)، أو الخاصة(مثل DB2 ,Sybase ,Oracle و Microsoft SQL Server) . المشاريع الحرة مثل Apache و لينكس و PostgreSQL لا يتم التحكم بها من قبل شركة واحدة، و لكن عن طريق مجتمع عالمي من المطورين و الشركات .

ملايين النسخ من PostgreSQL مثبتة على خوادم ويب و خوادم تطبيقات .

التقنية التي سننفذها هنا هي اقتراحات بسيطة، والتي نحاول استخدام أفضل أدوات التي اكتشفتها من خلال تعلمك في الفصول السابقة، وهي : هياكل البيانات عالية المستوى (القوائم والقواميس). والبرمجة الشيئية(بواسطة الكائنات). وغني عن القول أن أقوم بانتقاد نطاق واسع من الخيارات التي في هذا التمرين : يمكنك بالطبع علاج نفس المشاكل باستخدام طرق مختلفة.

هدفنا هو الحصول على عميل بدائي بسرعة، قادر على تواصل "حقيقي" مع خادم قاعدة البيانات. نحن نريد لعميلنا أن يبقى أداة صغيرة عامة جدا : يجب أن يكون قادرًا على إنشاء قاعدة بيانات صغيرة مع جداول متعددة، وإنتاج سجلات لكل واحدة، والسماح لنا باختبار نتائج الاستعلامات SQL الأساسية.

في الأسطر التالية، نحن نفترض أن لديك بالفعل وصول إلى خادم PostgreSQL، التي بها قاعدة بيانات "discotheque" التي تم صنعها من للمستخدم "jules" والذي كلمة مروره هي "abcde". هذا الخادم يمكن أن يتواجد على جهاز بعيد يمكن الوصول إليه عبر الشبكة، أو محليا على جهاز الحاسوب الخاص بك .

التكوين الكامل لخادم PostgreSQl هو خارج نطاق هذا الكتاب، لكن تثبيت أساسي ليس معقد على نظام تشغيل لينكس عن طريق توزيعة كلاسيكية مثل دبيان، أبنتو، ريد هات وسوزي ... يكفي تثبيت الحزمة التي تحتوي على الخادم (على سبيل المثال الحزمة **Postgresql-8.4** في النسخة الحالية لتوزيعة أبنتو في وقت كتابة هذه الأسطر)، ثم بتنفيذ العمليات القليلة التالية.

ادخل كمسؤول عن النظام لينكس (روت)، وقم بتعديل ملف التكوين **pg_hba.conf** الذي ينبغي أن يكون في الدليل **/etc/postgresql**ـ أو في **/var/lib/postgresql**.في هذا الملف، جميع أسطر التعليقات تبقى كما هي (و هذا معناه الأسطر التي تبدأ بالرمز #)، باستثناء ما يلي.

```
local    all    postgres    ident
local    all    all         md5
host     all    all    0.0.0.0    0.0.0.0    reject
```

بمساعدة الأمر (النظام) **sudo passwd**ـ يمكنك اختيار كلمة مرور للمستخدم postgres. وهذا المستخدم تم إنشاؤه تلقائيا أثناء تثبيت الحزمة، والتي سوف تكون رئيس كبير (أو postmaster) لخادم PostgreSQL الخاص بك.

أعد تشغيل خدمة PostgreSQL، وذلك باستخدام الأمر :

```
sudo  /etc/init.d/postgresql-8.4  restart
```

يجب عليك تسجيل دخول بعد ذلك إلى نظام لينكس كمستخدم postgres، (في البداية، هو الوحد القادر على صنع مستخدمين جدد لـ SGBDR)، وقم بتشغيل الأمر **createuser** :

```
createuser jules -d -P
Saisir le mot de passe pour le nouveau rôle : *****
Le saisir de nouveau : *****
```

```
Le nouveau rôle est-il super-utilisateur ? (o/n) n
Le nouveau rôle est-il autorisé à créer de nouveaux rôles ? (o/n) n
```

هذه الأوامر لتعريف مستخدم جديد "jules" لنظام PostgreSQL، وهذا المستخدم يمكن الإتصال بكلمة السر الخاصة به (في تمريننا، "abcde"). اسم المستخدم هو إجراء تعسفي : لا يتوافق بالضرورة مع اسم المستخدم المدرج بالفعل في نظام لينكس.

يمكنك الآن أستئناف الهوية المعتادة، وصنع قاعدة بيانات واحدا أو أكثر باسم "jules"، بمساعدة الأمر **createdb** :

```
createdb -U jules  discotheque
Mot de passe :  abcde
```

هذا يكفي في هذه المرحلة، الخادم PostgreSQL مستعد الآن للتفاعل مع عميل بيثون الذي سيتم شرحه في الصفحات القادمة .

وصف قاعدة بيانات في قاموس تطبيق

التطبيق الذي سيتفاعل مع قاعدة البيانات هو دائما تطبيق معقد تقريبا. فهو يحتوي بالضرورة على أسطر من التعليمات البرمجية، فمن الأفضل هيكلتها من خلال تجميعها في أصناف (أو على الأقل في دالات) مغلقة جيدا.

في أجزاء كثيرة من الكود، في كثير من الأحيان بعيدة جدا عن بعضا البعض ، يجب على كتل البيانات أن تأخذ بالاعتبار هيكل قاعدة البيانات، وهذا يعني، قطع (تقسيمها) إلى جداول وحقول، وكذلك إقامة علاقات التسلسل الهرمي في السجلات.

يبدوا أن التجربة تبين لك هيكل قاعدة البيانات النهائية. خلال التطوير، نحن ندرك أنه غالبا ما يكون ضروريا إضافة أو إزالة حقول، وفي بعض الأحيان تستبدل جدول مصمم بشكل سيئ بجدولين آخرين، إلخ. فإنه ليس من الحكمة برمجة أجزاء برمجية خاصة جدا لهيكل معينة، "من الصعب". أو بدلا من ذلك، من المستحسن للغاية وصف بنية كاملة من قاعدة البيانات في نقطة واحدة في البرنامج، ومن ثم استخدام هذا الوصف كمرجع لصنع نصف-آلي تعليمات محددة حول الجدول أو الحقل. هذا يتجنب، إلى حد كبير، كابوس الحاجة إلى تعقب وتعديل عدد كبير من التعليمات في الكود، في كل مرة هيكل قاعدة البيانات يتغير قليلا. وبدلا من ذلك، مجرد تغيير وصف فقط من المرجع، والجزء الأكبر من الكود لا يحتاج إلى تعديل .

> لدينا هنا فكرة واحدة لتحقيق تطبيقات قوية : ويجب دائما على برنامج معالج البيانات أن يكون مبنيا على أساس تطبيق قاموس .

ما نعنيه هنا أن "قاموس التطبيق" ليس بالضرورة أن يكون قاموس بيثون. أي بنية بيانات مناسبة يمكن تحويلها، والشيء المهم هو بناء مرجع مركزي واصفا البيانات التي تقترح على التعامل، مع ربما مجموعة من المعلومات حول التنسيق.

بسبب قدرتها على جمع: قوائم ومصفوفات مغلقة وقواميس تعمل لهذا العمل في كيان واحد من أي نوع. في المثال التالي في الصفحات القادمة، قمنا باستخدام قاموس، قيمه هي قوائم من مصفوفات مغلقة ولكن يمكنك اختيار تنظيم مختلف من نفس المعلومات.

لترسيخ كل هذا، لا يزال علينا حل سؤال مهم مثل: أين سنثبت هذا قاموس التطبيق ؟

و ينبغي النظر إلى المعلومات الخاصة من أي مكان في البرنامج. ولذلك فإن تثبيته داخل متغير عام شيء إلزامي، كبيانات أخرى لازمة لتشغيل جميع برنامجنا. أنت تعرف أنه ليس من المستحسن استخدام متغيرات عامة : لأنها تنطوي على مخاطر، تزداد مع زيادة حجم البرنامج. على أي حال، المتغيرات التي قلنا إنها عامة، لكنها في الحقيقة عامة داخل وحدة فقط. فإذا أردنا تنظيم برنامجنا على أنه مجموعة من الوحدات (و الذي هو ممارسة جيدة)، لن يكون لدينا الوصول للمتغيرات العامة الخاص بنا سوى بين بعضها.

لحل هذه المشكلة الصغيرة، يوجد حل بسيطة وأنيق : قم -في صنف معين- بجمع كافة المتغيرات التي تتطلب حالة عامة في التطبيق. ثم سوف نغلفها في مساحة أسماء الصنف، هذه المتغيرات يمكن استخدامها دون مشاكل في أي وحدة : يكفي أن يتم استدعاء الصنف. بالإضافة إلى ذلك، فإن استخدام هذه التقنية تنطوي على نتيجة مثيرة للاهتمام : الرمز "عام - global" هي متغيرات تم تعريفها بهذه الطريقة لتظهر بوضوح في اسمها المؤهل، لأن هذا الاسم يجب أن يبدأ بالصنف الذي يحتويه .

إذا اخترت على سبيل المثال، اسما وصفيا مثل **Glob** للصنف الهدف لاستيعاب متغيراتك "العالمية"، يجب عليك صنع مرجع لهذه المتغيرات جميعها في الكود مع أسماء وصفية مثل **Glob.cela** ، **Glob.ceci** ، إلخ.[91]

هذه التقنية التي سوف تكتشفها في الأسطر الأولى لسكريبتنا. لقد قمنا بتعريف صنف **Glob()**ـ التي ليس لديها سوى منشئ بسيط. ولن يتم تمثيل أي كائن من هذا الصنف، وفي الواقع لا تحتوي على أي أسلوب. متغيراتنا العامة سيتم تعرفهم كمتغيرات بسيطة للصنف، وحتى نتمكن من صنع مرجع لهم لبقية البرنامج كسمات لـ **Glob()**. اسم قاعدة البيانات، على سبيل المثال، يمكن العثور عليه في المتغير **Glob.dbName** ; اسم أو عنوان IP السيرفر موجود في المتغير **Glob.hos t**، إلخ:

```
1# class Glob(object):
2#     """Espace de noms pour les variables et fonctions <pseudo-globales>"""
3#
4#     dbName = "discotheque"      # اسم قاعدة البيانات
5#     user = "jules"              # المالك أو المستخدم
6#     passwd = "abcde"            # كلمة السر
7#     host = "127.0.0.1"          # اسم وعنوان IP للخادم
8#     port =5432
```

[91] يمكنك أيضا وضع متغيراتك "العامة" في وحدة تسمى Glob.py، ثم تقوم باستدعائها . إن استخدام وحدة أو صنف كمساحة للأسماء لتخزين متغيرات هي تقنية مماثلة تماما . إن استخدام صنف قد يكون أكثر مرونة و قابلية للقراءة، لأن يمكن أن تصاحب بقية السكريبت، ثم الوحدات هي بالضرورة في ملف منفصل.

```
    # هيكل قاعدة البيانات. قاموس من جداول وحقول :
    dicoT ={"compositeurs":[('id_comp', "k", "clé primaire"),
                            ('nom', 25, "nom"),
                            ('prenom', 25, "prénom"),
                            ('a_naiss', "i", "année de naissance"),
                            ('a_mort', "i", "année de mort")],
            "oeuvres":[('id_oeuv', "k", "clé primaire"),
                       ('id_comp', "i", "clé compositeur"),
                       ('titre', 50, "titre de l'oeuvre"),
                       ('duree', "i", "durée (en minutes)"),
                       ('interpr', 30, "interprète principal")]}
```

قاموس التطبيق يصف هيكل قاعدة البيانات التي تحتوي داخل المتغير **Glob.dicoT**.

في هذا القاموس، يوجد مفاتيح وأسماء الجداول. أما القيم، فكل منها عبارة عن قائمة تحتوي على وصف جميع الحقول(مجالات) في الجدول، على شكل مصفوفات مغلقة.

كل مصفوفة مغلقة Tuple تصف حقلا معينا في الجدول. لتجنب تبعثر تمريننا، سوف نحدد هذا الوصف إلى ثلاثة معلومات فقط : اسم الحقل ونوعه وتعليق قصير.

في التطبيقات الحقيقية، فإنه سيتم وضع المزيد من المعلومات هنا، على سبيل المثال للقيم الحد التي لأي حقل حد من البيانات، وتنسيق تطبيق عندما يتم عرضه على الشاشة أو لطباعته، والنص يجب وضعه في الجزء الأعلى من العمود عندما نريد تقديمه في جدول، إلخ.

قد يبدو مملا جدا وصف هيكل البيانات بالتفصيل ، لذلك سوف تبدأ على الفور بالتفكير حول مختلف الخوارزميات التي يجب أن يتم تنفيذها من أجل معالجتها. والتي يجب أن يتم القيام بها بشكل جيد، وهذا الوصف المنظم سوف يوفر لك الكثير من الوقت في وقت لاحق، لأنه سوف يسمح لك d'automatiser العديد من الأشياء. سوف ترى بعد قليل. بالإضافة إلى ذلك، يجب أن تقنع نفسك أن هذه المهمة الصعبة تؤهلك لهيكلة بشكل صحيح عملك : تنظيم النماذج والاختبارات وإلخ .

تعريف صنف كائنات-واجهة

الصنف **Glob()** (تم وصفه سابقا) سيتم تثبيته في بداية السكربت، أو في وحدة منفصلة يتم استدعاؤها في بداية السكربت. وللبقية، سنفترض أنه يستخدم الصيغة الأخيرة : سوف نقوم بحفظ الصنف **Glob()** في وحدة تسمى **dict_app.py**، حيث يمكننا استدعاؤها في السكربت التالي.

هذا السكريبت الجديد يعرف صنف كائنات-الواجهة. في الوقع نحن نحاول الاستفادة مما تعلمناه في الفصول السابقة، وبالتالي التركيز على البرمجة الشيئية(الكائنات)، لصنع قطعة من التعليمات البرمجية المغلقة والقابلة للاستخدام على نطاق واسع.

كائنات-الواجهات التي تريد صنعها ستكون مشابهة لكائنات-الملفات التي استخدمناها لمعالجة الملفات في الفصل 9. أنت تتذكر على سبيل المثال أننا نفتح الملف عن طريق إنشاء كائن-ملف، باستخدام دالة-الصنع **open()**. بطريقة مماثلة، سوف نقوم بفتح التواصل مع قاعدة البيانات، نبدأ بصنع كائن-واجهة باستخدام الصنف **GestionBD()**، والذي سيقوم بتأسيس الاتصال. للقراءة أو للكتابة في ملف مفتوح، سوف نستخدم مختلف أساليب كائن-الملف. على نحو مماثل، سوف نجعل عملياتنا على قاعدة البيانات من خلال أساليب مختلفة لكائن-الواجهة .

```
import sys
from pg8000 import DBAPI
from dict_app import *

class GestionBD(object) :
    """Mise en place et interfaçage d'une base de données PostgreSQL"""
    def __init__(self, dbName, user, passwd, host, port =5432):
        "Établissement de la connexion - Création du curseur"
        try:
            self.baseDonn = DBAPI.connect(host =host, port =port,
                                          database =dbName,
                                          user=user, password=passwd)
        except Exception as err:
            print('La connexion avec la base de données a échoué :\n'\
                  'Erreur détectée :\n%s' % err)
            self.echec =1
        else:
            self.cursor = self.baseDonn.cursor()   # صنع مؤشر
            self.echec =0

    def creerTables(self, dicTables):
        "Création des tables décrites dans le dictionnaire <dicTables>."
        for table in dicTables:               # تدوير مفاتيح القاموس
            req = "CREATE TABLE %s (" % table
            pk =''
            for descr in dicTables[table]:
                nomChamp = descr[0]           # تسمية حقل الذي تريد إنشاءه
                tch = descr[1]                # نوع الحقل الذي تريد إنشاءه
                if tch =='i':
                    typeChamp ='INTEGER'
                elif tch =='k':
                    # حقل 'مفتاح الأساسي' (عدد صحيح لزيادة تلقائيا)
                    typeChamp ='SERIAL'
                    pk = nomChamp
                else:
                    typeChamp ='VARCHAR(%s)' % tch
                req = req + "%s %s, " % (nomChamp, typeChamp)
            if pk == '':
                req = req[:-2] + ")"
            else:
                req = req + "CONSTRAINT %s_pk PRIMARY KEY(%s))" % (pk, pk)
            self.executerReq(req)

    def supprimerTables(self, dicTables):
        "Suppression de toutes les tables décrites dans <dicTables>"
        for table in list(dicTables.keys()):
            req ="DROP TABLE %s" % table
            self.executerReq(req)
        self.commit()                          # نقل -> القرص
```

```
50#
51#     def executerReq(self, req, param =None):
52#         "Exécution de la requête <req>, avec détection d'erreur éventuelle"
53#         try:
54#             self.cursor.execute(req, param)
55#         except Exception as err:
56#             # عرض استعلام ورسالة خطأ النظام :
57#             print("Requête SQL incorrecte :\n{}\nErreur détectée :".format(req))
58#             print(err)
59#             return 0
60#         else:
61#             return 1
62#
63#     def resultatReq(self):
64#         "renvoie le résultat de la requête précédente (une liste de tuples)"
65#         return self.cursor.fetchall()
66#
67#     def commit(self):
68#         if self.baseDonn:
69#             self.baseDonn.commit()         # نقل المؤشر -> القرص
70#
71#     def close(self):
72#         if self.baseDonn:
73#             self.baseDonn.close()
```

تعليقات

• الأسطر من 1 إلى 3 : بالإضافة إلى وحدة **dict_app** التي تحتوي على متغيرات "العامة"، قمنا باستدعاء وحدة **sys** التي تضم بعض دالات النظام، وخاصة وحدة **pg8000** التي تشمل كل ما يلزم للتواصل مع PostgreSQL. هذه الوحدة ليست جزءا من بيثون القياسية. بل هي وحدة واجهة بيثون-PostgreSQL متوفرة بالفعل لبيثون 3. العديد من المكتبات الأخرى العديدة أكثر كفاءة، متوفر منذ مدة طويلة للإصدارات السابقة من بيثون، سوف تتكيف بالتأكيد (السائق الممتاز **psycopg2** سوف يكون جاهزا قريبا).

> لتثبيت *pg8000*، انظر إلى صفحة 413.

• السطر 7 : عند إنشاء كائنات-الواجهة، سوف نوفر برامترات الاتصال : اسم قاعدة البيانات، اسم المستخدم، الاسم أو عنوان IP للجهاز الذي يقع الخادم. رقم منفذ الاتصال عادة ما يكون افتراضيا. يفترض أن تكون كل هذه المعلومات لديك.

• الأسطر من 9 إلى 19 : من المستحسن للغاية وضع تعليمات برمجية لإجراء إتصال داخل معالج الاستثناء **try-except-else** (انظر للصفحة 125)، لأننا لا نستطيع أن نفترض أن الخادم سيكون يمكن الوصول إليه ضروريا. لاحظ أن الأسلوب **__init__()** لا يمكنه أن يرجع قيمة (باستخدام التعليمة **return**)، لأنها يتم استدعاؤها تلقائيا بواسطة بيثون عندما تقون بتمثيل الكائن. في الواقع : ما يتم إرجاعه في هذا البرنامج المستدعي هو كائن بني حديثا. وبالتالي ليس هنالك تقرير نجاح أو فشل الاتصال ببرنامج الاستعداد باستخدام قيمة رجوع. وهنالك حل بسيط لهذه المشكلة

الصغيرة لتخزين نتيجة محاولة الاتصال في سمة مثيل (المتغير **self.echec**)، ويمكن للبرنامج الذي تم استدعاؤه اختيار ما quand bon lui semble .

- الأسطر من 21 إلى 42 : هذا الأسلوب automatise صنع جميع الجداول لقاعدة البيانات، للاستفادة من وصف قاموس التطبيق، يجب عليك تمرير برامتر. وسوف تكون هذه الآلية أكثر أهمية من الواضح، لأن هيكل قاعدة البيانات سوف تكون أكثر تعقيدا (تخيل على سبيل المثال قاعدة بيانات تحتوي على 35 جدول !). حتى لا نعقد الإثبات، سوف نستخدم هذه القدرة لهذا الأسلوب لصنع حقول من أنواع integer و varchar. لا تتردد في إضافة تعليمات لإنشاء حقول من أنواع أخرى.

إذا أردت تفصيل الكود، سوف تجد أنه ببساطة يمكنك بناء استعلام SQL لكل جدول، قطعة قطعة، في سلسلة نصية **req**. ثم سوف يتم تمرير للأسلوب **executerReq()** لتنفيذه. فإذا كنت ترغب في عرض الاستعلام ثم بنائها، يمكنك بالطبع إضافة تعليمة `print(req)` مباشرة بعد السطر 42.

يمكنك أيضا أن تضيف للأسلوب قدرة تنفيذ قيود التكامل المرجعي، على أساس مكمل للقاموس التطبيق الذي يصف هذه القيود. نحن لا نطور هذه المشكلة هنا، لكن لا ينبغي vous poser de problème si vous savez de quoi il retourne.

- الأسطر من 44 إلى 49 : أبسط كثيرا من سابقتها، هذا الأسلوب يستخدم نفس مبدأ لحذف جميع الجداول الموضحة في قاموس التطبيق.
- الأسطر من 51 إلى 61 : هذا الأسلوب يوجه ببساطة استعلام كائن المؤشر. فائدته هي تبسيط الوصول إليها وإنتاج رسالة خطأ إذا لزم الأمر.
- الأسطر من 63 إلى 73 : هذه الأساليب ليست سوى تتابع بسيط لكائنات التي يتم صنعها من وحدة **pg8000** : كائن-الاتصال ينتج بواسطة دالة-الصنع **DBAPI.connect()**، وكائن المؤشر المطابق. يسمح لهم بتبسيط الكود قليلا للبرنامج الذي استدعاه .

بناء نموذج مولد

لقد قمنا بإضافة هذا الصنف لتمريننا لشرح كيف يمكنك استخدام نفس قاموس التطبيق لتطوير كود المولد. الفكرة هنا هي تحقيق صنف كائنات-أشكال قادرة على دعم ترميز السجلات أي الجداول ، ببناء تعليمات الإدخال المناسبة تلقائيا باستخدام المعلومات من قاموس التطبيق.

في تطبيق حقيقي، يجب أن يكون هذا النموذج مبسطا ومعدلا بشكل كبير، وسوف يكون على الأرجح على شكل نافذة متخصصة، بها حقول الإدخال والملاصق الخاص التي تتولد تلقائيا. نحن لا ندعي أنه ليس مثالا جيدا، لكننا نريد فقط أن نظهر لك كيف يمكنك automatiser منشئه إلى حد كبير. حاول القيام بنماذجك الخاصة باستخدام مبادئ مماثلة .

```
1#  class Enregistreur(object):
2#      """classe pour gérer l'entrée d'enregistrements divers"""
3#      def __init__(self, bd, table):
4#          self.bd =bd
5#          self.table =table
6#          self.descriptif =Glob.dicoT[table]   # descriptif des champs
7#
8#      def entrer(self):
9#          "procédure d'entrée d'un enregistrement entier"
10#         champs ="("           # مسودة سلاسل لأسماء الحقول
11#         valeurs =[]           # قائمة للقيم المقابلة
12#         # طلب قيمة على التوالي لكل حقل :
13#         for cha, type, nom in self.descriptif:
14#             if type =="k":    # نحن لن نطلب رقم تسجيل
15#                 continue      # (ترقيم تلقائي) من المستخدم)
16#             champs = champs + cha + ","
17#             val = input("Entrez le champ %s :" % nom)
18#             if type =="i":
19#                 val =int(val)
20#             valeurs.append(val)
21#
22#         balises ="(" + "%s," * len(valeurs)       # علامات التحول
23#         champs = champs[:-1] + ")"    # حذف الفاصلة الأخيرة,
24#         balises = balises[:-1] + ")"  # وإضافة قوس
25#         req ="INSERT INTO %s %s VALUES %s" % (self.table, champs, balises)
26#         self.bd.executerReq(req, valeurs)
27#
28#         ch =input("Continuer (O/N) ? ")
29#         if ch.upper() == "O":
30#             return 0
31#         else:
32#             return 1
```

تعليقات

- الأسطر من 1 إلى 6 : في وقت التمثيل، كائنات من هذا الصنف سوف يتلقى مرجع واحد من جداول القاموس. هذا ما يتيح لهم الوصول إلى وصف الحقول.
- السطر 8 : الأسلوب **entrer()** يولد نموذج نفسه. وهو يدعم سجلات الإدخال في الجداول، من خلال التكيف مع هيكل الخاص بها من خلال وصف الموجود في القاموس. وظيفتها عملية بينة مرة أخرى قطعة قطعة سلسلة نصية التي ستصبح استعلام SQL، كما في الأسلوب **creerTables()** للصنف **GestionBD()** الموضح سابقا. يمكنك بالطبع إضافة أساليب أخرى إلى هذا الصنف، لمعالجة على سبيل المثال حذف وأو تعديل السجلات .

- الأسطر من 12 إلى 20 : سمة المثيل **self.descriptif** تحتوي على قائمة ومصفوفات مغلقة، وكل واحد منها يتكون من 3 عناصر، وهي اسم الحقل ونوع البيانات التي سوف يتلقاها، والوصف "بوضوح" . الحلقة **for** في السطر 13 تقوم بتكرار هذه القائمة وعرض لكل حقل رسالة موجهة على أساس الوصف الذي يصاحب هذا الحلق. عندما يقوم المستخدم بإدخال القيم المطلوبة، سبتم حفظها في قائمة المنشئ، في حين يتم إضافة اسم الحقل إلى تنسيق السلسلة.
- الأسطر من 22 إلى 26 : عندما يتم تكرار جميع الحقول، يتم تجميع الاستعلام وتنفيذه. كما شرحنا في الصفحة 309، لا ينبغي أن تكون القيم المدرجة في سلسلة الاستعلام نفسها، لكن تم تمريرها كبرامتر للأسلوب **execute()**.

جسم التطبيق

لا يبدو مفيدا تطوير المزيد من هذا التمرين في الإطار اليدوي للمشروع. إذا كنت مهتما، يجب أن نعرف الآن ما يكفي بدء أي تجاربك الشخصية. يرجى الإطلاع كتب مراجع، كما على سبيل المثال : How to program - كيف تبرمج لـ Deitel و coll، أو مواقع متخصصة لملحقات بيثون.

السكريبت التالي هو تطبيق صغير مصمم لاختبار أصناف تم وصفها في الصفحات السابقة. لا تتردد في تحسينها، ثم اكتب واحدة أخرى مختلفة تماما !

```
###### البرنامج الرئيسي : #########

# صنع كائن واجهة مع قاعدة البيانات :
bd = GestionBD(Glob.dbName, Glob.user, Glob.passwd, Glob.host, Glob.port)
if bd.echec:
    sys.exit()

while 1:
    print("\nQue voulez-vous faire :\n"\
          "1) Créer les tables de la base de données\n"\
          "2) Supprimer les tables de la base de données ?\n"\
          "3) Entrer des compositeurs\n"\
          "4) Entrer des oeuvres\n"\
          "5) Lister les compositeurs\n"\
          "6) Lister les oeuvres\n"\
          "7) Exécuter une requête SQL quelconque\n"\
          "9) terminer ?                       Votre choix :", end=' ')
    ch = int(input())
    if ch ==1:
        # صنع جميع الجداول الموصوفة في القاموس :
        bd.creerTables(Glob.dicoT)
    elif ch ==2:
        # إزالة صنع جميع الجداول الموصوفة في القاموس :
        bd.supprimerTables(Glob.dicoT)
    elif ch ==3 or ch ==4:
        # صنع <enregistreur> لملحنين أو :
        table ={3:'compositeurs', 4:'oeuvres'}[ch]
        enreg =Enregistreur(bd, table)
        while 1:
            if enreg.entrer():
                break
    elif ch ==5 or ch ==6:
        # قائمة كل الملحنين, ou toutes les oeuvres :
        table ={5:'compositeurs', 6:'oeuvres'}[ch]
        if bd.executerReq("SELECT * FROM %s" % table):
            # تحليل نتيجة الإستعلام أدناه :
            records = bd.resultatReq()      # سيكون نفق من المصفوفات المغلقة
            for rec in records:             # => كل سجل
                for item in rec:            # => كل حقل للسجل
                    print(item, end=' ')
                print()
    elif ch ==7:
        req =input("Entrez la requête SQL : ")
        if bd.executerReq(req):
            print(bd.resultatReq())          # سيكون نفق من المصفوفات
    else:
        bd.commit()
        bd.close()
        break
```

تعليقات

- يجب أن نفترض بالطبع أن الأصناف المذكورة أعلاه موجودة في نفس السكريبت، أو تم استدعاءه.
- الأسطر من 3 إلى 6 : يتم إنشاء كائن-الواجهة هنا. فإذا فشل، سمة المثيل **bd.echec** ستكون قيمتاه 1. وسطور 5 و 6 يسمحون بإغلاق التطبيق فورا (الدالة **exit()** لوحدة **sys** تستخدم خصيصا لهذا).
- السطر 8 : ما تبقى من التطبيق هو تقديم نفس القائمة باستمرار إلى أن يختار المستخدم الخيار رقم 9.
- الأسطر 27 و 28 : الصنف **Enregistreur()** يقبل معالجة السجلات من أي جدول ، لتحديد أين يجب أن تستخدم عند التمثيل، استخدم قاموس صغير الذي يشير إلى اسم الحفظ، وهذا يتوقف على الاختيار الذي أدلى به المستخدم (الخيار رقم 3 أو رقم 4).
- الأسطر من 29 إلى 31 : الأسلوب **entrer()** لكائن-الحافظ يقوم بإرجاع القيمة 0 أو 1 اعتمادا إذا كان اختيار المستخدم مواصلة إدخال السجلات، أو التوقف. إن اختبار هذه القيم تسمح بوقف حلقة التكرار وفقا لذلك.
- الأسطر من 35 إلى 44 : يقوم الأسلوب **executerReq()** بإرجاع قيمة 0 أو 1 اعتمادا على ما إذ تم قبول الاستعلام أو لا من قبل الخادم. يمكننا اختبار هذه القيمة لنقرر إذا كان الناتج يجب عرضه أو لا .

تمارين

2.16 قم بتعديل السكريبت الموضح في هذه الصفحات من أجل إضافة جدول إضافي إلى قاعدة البيانات. يمكن أن يكون على سبيل المثال "orchestres - عصابات"، الذي يحتوي كل سجل منها على اسم المجرم وزعيمه والعدد الإجمالي للصكوك .

3.16 قم بتعديل أنواع أخرى من حقو إلى أحد الجداول (على سبيل المثال حقل من نوع حقيقي أو نوع date - تاريخ)، وقم بتغيير السكريبت وفقا لذلك .

17

تطبيقات الوب

ربما قد تعلمت سابقا أشياء كثيرة عن كتابة صفحات الويب .هل تعلم أن هذه الصفحات هي عبارة عن مستندات مكتوبة بلغة *HTML* والتي يمكن الوصول إليها عن طريق الإنترنت باستخدام برامج خاصة تسمى بالمتصفحات (مثل فاير فوكس وإنترنت إكسبلورر وسفاري وأوبرا وغاليون وكونكيورر ...) .يتم تثبيت صفحات هوتميل في دلائل عامة (فهرسات عامة) من حاسوب آخر يشغل بشكل مستمر برنامج يدعى خادم الويب (مثل أباتشي ولايت هتتبد وزيتامي وإإس) وعندما يتم تأسيس اتصال بين الحاسوب وحاسوبك ،يمكن للمتصفح التفاعل مع برنامج الخادم عن طريق إرسال طلبات (عن طريق مجموعة متنوعة من الأجهزة والبرمجيات التي لن تتم مناقشتها هنا مثل : خطوط الهاتف والموجهات (الراوترات) وبروتوكولات الإتصال ...) ويعرض المتصفح النتائج في إستجابة لهذه الطلبات .

صفحات ويب تفاعلية

بروتوكول http الذي يدير الانتقال بين صفحات الويب يتيح تبادل البيانات في كلا الاتجاهين. ولكن إذا كان الموقع بسيطا (لا تتفاعل مع الموقع من خلال إدخال أشياء إلخ)يقام نقل البيانات في اتجاه واحد من اثنين ، وهو الخادم إلى المتصفح : النصوص والصور والملفات المختلفة التي يتم إرسالها إلى المتصفح بأعداد كبيرة (هذه هي الصفحات التي يتم تصفحها) ومع ذلك ، يرسل المتصفح إلى الخادم كميات ضئيلة جدا من المعلومات إلى الخادم : في الأساس عناوين الصفحات التي يرغب المستخدم في التفاعل معها.

و أنت تعرف أن هنالك العديد من المواقع حيث يطلب منك المزيد من المعلومات (أن تتفاعل أكثر) مثل : مراجع الشخصية للتسجيل في النادي أو حجز في فندق ، ومن المعلومات التي سترسلها هي مثلا : رقم هاتفك أو رقم بطاقة الائتمان لطلب بند ما على موقع للتجارة الإلكترونية أو أرائك أو اقتراحاتك ، إلخ ...

في هذه الحالات ، يمكنك أن تتخيل أن المعلومات المعتمدة ستنقل إلى جانب الخادم من خلال برنامج محدد. بالتالي يجب ربطها بهذا البرنامج عن بعد في الخادم.

أما بالنسبة لصفحات الويب المصممة لاستيعاب هذه المعلومات (وتسمى النماذج)، وسوف توفر لهم ترمين الحاجيات المختلفة (حقول الإدخال، خانات ومربعات القوائم، الخ.)، بحيث يمكن للمتصفح تقديم طلب إلى خادم جنبا إلى جنب مع وسائط.

إذا يمكن للخادم التعامل مع هذه المدخلات ببرنامج معالج (يعني يتعامل مع المدخلات) خاص واعتمادا على هذه النتائج يرسل البرنامج جواب مناسب للمستخدم تحت شكل صفحة ويب جديد (معناه الانتقال إلى صفحة أخرى بها الإجابة مثل صفحة تم إنهاء التسجيل).

هنالك طرق مختلفة لتقوم بعمل مثل هذه البرامج الخاصة والتي نسميها الآن تطبيقات الويب.

واحدة من الطرق الأكثر شعبية في الوقت الحاضر هو استخدام صفحات HTML "تم إثراؤها" باستخدام برامج نصية مكتوبة (السكريبتات) بمساعدة لغات برمجة معينة مثل بي أتش بي. يتم إدراج هذه السكريبتات مباشرة في شيفرة ال HTML بين وسوم خاصة وسيتم تنفيذها من قبل خادم الويب (و على سبيل المثال أباتشي) على أن يتم تجهيز هذا الكود مع وحدة المترجم المناسبة. يمكنك أن تفعل ذلك مع بيثون عبر صيغة معدلة بشكل طفيف من لغة تسمى PSP (صفحات خادم بيثون).

هذا المنهج لديه عيب وهو خلط أكواد ال HTML ومن هذه الخلوطات(جمع كلمة خلط) التلاعب بخلط أجزاء من سكريبتات بي أتش بي أو بي أس بي داخل الوسوم مما يعرض لمشاكل عند القراءة بشكل عام.

و أفضل أسلوب هو العمل (بشكل منفصل) على كتابة النصوص التي تولد كود HTML كلاسيكية في شكل سلاسل وتوفير وحدة (موديل) على خادم الويب لتفسير هذه النصوص وتعود بكود HTML ردا على استفسارات المتصفح (على سبيل المثال **mod_python**، في حالة أباتشي).

في حالة أباتشي)

و لكن مع بيثون ،يمكننا دفع هذا النوع من المنهج إلى أبعد من ذلك من خلال التطوير بأنفسنا لخادم ويب حقيقي متخصص ، مستقل تماما ، في واحدة تحتوي على وظائف برامج الخاصة مطلوبة لتطبيقنا. نستطيع القيام بذلك مع بيثون لأنها بنيت كافة المكتبات اللازمة لإدارة بروتوكول HTTP إلى اللغة. على هذا الأساس قد أنتج العديد من المبرمجين المستقلين أيضا وأتاحوا للمجتمع مجموعة من الأدوات التطوير لتسهيل تطوير مثل هذه التطبيقات الويب. في الفترة المتبقية من دراستنا ، سوف نستخدم واحدة منهم. اخترنا CherryPy لأنه يبدو جيدا ومناسبا مع أهداف هذا العمل .

مهم

الذي سنشرحه في الفقرات التالية سيكون عمليا مباشرة على الإنترنت من مدرستك أو في شركتك. فيما يتعلق بسلبيات الأنترنت الأمور القليلة أكثر تعقيدا ، بالإضافة إلى أن تثبيت البرنامج على حاسوب خادم (سيرفر) متصل بشبكة الأنترنت لا يمكن أن يتم إلا بموافقة مالكها. إذا كان المزود قد أتاح لك مساحة حيث يسمح لك بتثبيت صفحات ويب ساكنة (وهذا يعني بعض الوثائق البسيطة للاستشارة) هذا لا يعني أنك لن تكون قادرا على تشغيل برامج ! من أجل تشغيلها سيكون من الضروري أن تحصل على تصريح وعدد من المعلومات إلى المزود .معظمهم يرفضون السماح لتثبيت تطبيقات مستقلة عن النوع الذي وصفناه أدناه ، ولكن يمكنك بسهولة تخويلها بحيث تكون صالحة للاستخدام أيضا مع ***mod_python*** لأباتشي وهي عموما موجودة[92].

خادم ويب في بيثون نقية

لقد أصبح الاهتمام في تطوير الشبكة مهم جدا في عصرنا الحاضر، وهناك طلب قوي على واجهات البرمجة وبيئات مناسبة تماما لهذه المهمة. لكن حتى لو أنه لا يمكن مطالبة لغات عالمية مثل C / C + +، وبالفعل بيثون على نطاق واسع في جميع أنحاء العالم لكتابة البرامج الطموحة، بما في ذلك مجال خوادم التطبيقات على شبكة الإنترنت. وقد اجتذبت متانة وسهولة التنفيذ من اللغة للمطورين العديد من الموهوبين الذين جعلوا من أدوات تطوير مواقع الويب إلى أعلى مستوى. قد يكون العديد من هذه التطبيقات تهمك إذا كنت تريد أن تفعل ذلك بنفسك ومواقع الويب التفاعلية من مختلف الأنواع.

المنتجات الحالية هي في معظمها برمجيات الحرة. ويمكن أن تغطي مجموعة واسعة من الاحتياجات، بدءاً من موقع شخصي صغير من بضع صفحات وحتى موقع تجاري كبير ، وقادرا على الإجابة على آلاف الطلبات يوميا، ومختلف القطاعات التي تدار من قبل أشخاص من دون تدخل من مختلف المهارات (مصممي الرسوميات والمبرمجين، وقاعدة بيانات، الخ ...).

الأكثر شهرة من هذه المنتجات هو برنامج Zope، التي سبق اعتمادها من قبل كبرى المنظمات الخاصة والعامة لتطوير الشبكات الداخلية والخارجية التعاونية. هذا هو في الواقع خادم تطبيق النظام، والأداء العالي، وهي آمنة بشكل كامل تقريبا ومكتوبة في بيثون، ويمكن أن تدار عن بعد باستخدام واجهة ويب بسيطة. فمن غير الممكن وصفنا استخدام Zope في هذه الصفحات: هذا الموضوع واسع جدا، والكتاب بأكمله لا يكفي. كن على علم بأن هذا المنتج قادر تماما على التعامل مع مواقع الشركات الكبيرة جدا وتوفير مزايا كبيرة على حلول أفضل معروفة مثل PHP أو جافا.

من الأدوات أخرى المتاحة أقل طموحا ولكن مثيرة للاهتمام أيضا. مثل Zope، معظمها يمكن أن تحملهم مجانا من الإنترنت. وفي الحقيقة هي مكتوبة في بيثون ويمكنك استخدامها على عدة أنظمة ويندوز أو لينكس أو ماك على حد سواء. ويمكن

[92] يرجي الرجوع إلى أعمال أكثر تخصصا مثل Cherrypy Essentials, par Sylvain Hellegouarch, Packt Publishing, Birmingham, 2007. كتاب مرجعي عن Cherrypy .

استخدام كل بالتزامن مع خادم ويب "تقليدي" مثل أباتشي أو Xitami (وهو أيضا من خيار جيد إذا كان الغرض من الموقع تحقيق حمولة من الروابط المهمة على شبكة الإنترنت)، ولكن معظمها تحتوي على الخادم الخاص بها، مما يتيح لها العمل جيدا بشكل مستقل تماما. هذا الاحتمال مثير للاهتمام بشكل خاص في تطوير الموقع، لأنه يسهل استكشاف الأخطاء وإصلاحها.

الحكم الذاتي الكامل وسهولة التنفيذ جعل هذه المنتجات حلولا جيدة لإيجاد مواقع الإنترنت المتخصصة، بما في ذلك الشركات الصغيرة والمتوسطة، أو الحكومات، أو المدارس. إذا كنت ترغب في بناء تطبيق بيثون يتم الوصول إليها عن بعد عبر متصفح الإنترنت، فهذه الأدوات قد صنعت لك. هناك مجموعة واسعة: جانغو، Turbogears، أبراج، Karrigell، SPYCE، CherryPy، Webware، كيخوته، ملتوية، إلخ.[93]. اختار وفقا لحاجاتك، وسوف يكون لك مجموعة كبيرة للاختيار منها.

في ما يلي، ونحن نصف خطوة خطوة لتطوير تطبيق ويب يشغل باستخدام CherryPy. الذي يمكنك أن تجده في موقع : http://www.cherrypy.org بطريقة ودية للغاية، لأنه يسمح له بتطوير موقع على شبكة الانترنت كتطبيقات بيثون التقليدية، واستنادا إلى مجموعة من الكائنات. تولد كود HTML استجابة لطلبات HTTP الموجهة إليها عبر أساليبها، وينظر إلى هذه الأساليب بعناوين معتادة من قبل المتصفحات.

> لبقية هذا النص، فإننا نفترض أن لديك بعض من أساسيات *HTML*. ونحن نفترض أيضا أنه تم تثبيت مكتبة *CherryPy* على محطة العمل الخاصة بك (تم وصف هذا التبيث في المرفق صفحة413).

أول مشروع: إطلاق الموقع على شبكة الإنترنت مع الحد الأدنى من مكونات الصفحة.

في مجلد العمل الخاص بك، اصنع ملفا نصيا صغيرا وسمّه **tutoriel.conf**، ويحتوي على التالي:

```
[global]
server.socket_host = "127.0.0.1"
server.socket_port = 8080
server.thread_pool = 5
tools.sessions.on = True
tools.encode.encoding = "Utf-8"
[/annexes]
tools.staticdir.on = True
tools.staticdir.dir = "annexes"
```

هذا ملف بسيط للإعدادات يجعل خادم الويب CherryPy لدينا يتشاور عند بدء التشغيل. لاحظ رقم المنفذ (8080 في مثالنا). ولعلكم تعلمون أن المتصفحات برامج تنتظر لتجد الخدمات على شبكة الإنترنت من المنفذ 80 افتراضيا. إذا كنت صاحب الجهاز، ولا تريد تثبيت أي برنامج خادم ويب آخر، لذلك ربما كان من الأفضل أن تحل محل 8080 بـ 80 في ملف التكوين هذا : والمتصفحات التي سيتم الاتصال بملقم يجب أن يتم تحديد رقم منفذ في العنوان. ومع ذلك، إذا كنت تفعل هذه

[93] في وقت كتابة هذه السطور، CherryPy أتيح للتو نسخة 3 من بيثون . من بين الأدوات الأخرى المذكورة هنا، العديد منها يجري تكييفها، لكنها على أي حال قابلة للاستخدام تماما مع الإصدارات السابقة من بيثون .

التمارين على جهاز لم تكن أنت المسؤول عنه ، يعني لم يكن لديك الحق في استخدام أرقام المنافذ مثل 1024 ((لأسباب أمنية). في هذه الحالة، يجب استخدام رقم منفذ أعلى من 80ـ مثل التي وضعناها. وهذا ينطبق حتى لو كان خادم الويب آخر (اباتشي وعلى سبيل المثال) قيد التشغيل على الجهاز الخاص بك، لأن هذا البرنامج يستخدم على الأرجح المنفذ 80، بالفعل افتراضيا.

السطر **server.thread_pool =5** يشير إلى عدد من المواضيع خادم CherryPy ستفتح بالتوازي لمعالجة طلبات من نفس الوقت لمختلف المستخدمين. المواضيع هي "ابن" من التنفيذ المتزامن للبرنامج: سنضع شرحها في الفصل 19.

لاحظ أيضا خط الترميز. هذا هو الترميز الذي سوف نتستخدم في CherryPy في إنتاج صفحات الويب. فمن الممكن بعض المتصفحات الأخرى نتوقع أن UTF-8 هو الترميز الافتراضي. اذا كنت تحصل على أحرف غير صحيحة في المتصفح الخاص بك (في بعض الأحيان على شكل مربعات) عندما تواجهك هذه المشكلة الموضحة أدناه، وكرر المحاولة عن طريق تحديد الترميزات المختلفة في هذا الخط.

الأسطر الثلاثة الأخيرة في الملف تشير إلى مسار الدليل حيث تقوم بوضع 'ثابت' الوثائق التي قد يحتاج موقعك (صور، الشكل، الخ.).

الآن اكتب ملف السكريبت أدناه :

```
import cherrypy

class MonSiteWeb(object):          # الصنف السيد للتطبيق
    def index(self):               # أسلوب يتم إستدعائه كجذر (/)
        return "<h1>Bonjour à tous !</h1>"
    index.exposed = True           # الأسلوب يجب أن يكون "منشور'
        
###### البرنامج الرئيسي : #############
cherrypy.quickstart(MonSiteWeb(), config ="tutoriel.conf")
```

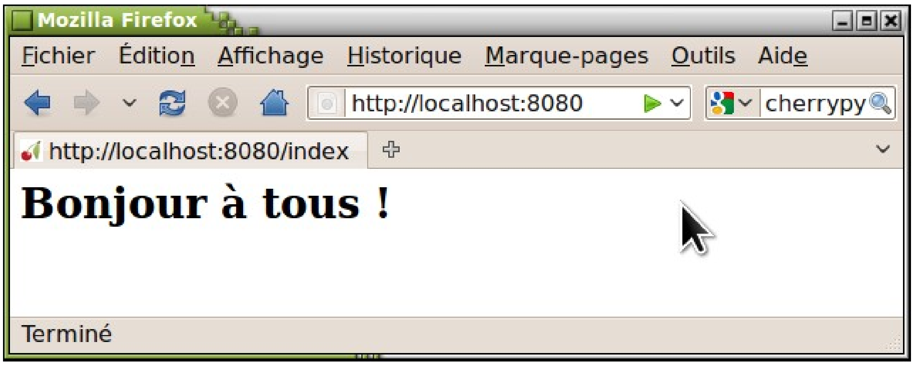
شغل البرنامج النصي(السكريبت). إذا كان كل شيء على ما يرام، ستحصل على بعض أسطر من معلومات مشابهة لما يلي في طرفيتك. فإنها تؤكد أن "شيئا ما" بدأ، وينتظر الأحداث:

```
[07/Jan/2010:18:00:34] ENGINE Listening for SIGHUP.
[07/Jan/2010:18:00:34] ENGINE Listening for SIGTERM.
[07/Jan/2010:18:00:34] ENGINE Listening for SIGUSR1.
[07/Jan/2010:18:00:34] ENGINE Bus STARTING
[07/Jan/2010:18:00:34] ENGINE Started monitor thread '_TimeoutMonitor'.
[07/Jan/2010:18:00:34] ENGINE Started monitor thread 'Autoreloader'.
[07/Jan/2010:18:00:34] ENGINE Serving on 127.0.0.1:8080
[07/Jan/2010:18:00:34] ENGINE Bus STARTED
```

كانت في الواقع مجرد أسطر لبدء خادم الويب!

سوف نتأكد فقط من أنها تعمل بشكل جيد، وذلك باستخدام متصفحك المفضل. إذا كنت تستخدم هذا المتصفح على نفس الجهاز كملقم، أشر نحو عنوان مثل http://localhost:8080، إن **localhost** هو مصطلح يستخدم لوصف الجهاز المحلي (يمكنك

أيضا تحديد ذلك باستخدام عناوين IP التقليدية: 127.0.0.1)، و 8080 رقم المنفذ المحدد من ملف configuration[94]. يجب أن تشاهد الصفحة الرئيسية التالية:



يمكنك أيضا الوصول إلى نفس الصفحة الرئيسية من جهاز آخر، عن طريق توفير عنوان IP لمتصفحه أو اسم السيرفر الخاص بك على الشبكة المحلية، بدلا من `localhost`.

دعونا الآن نجرب السكريبت الخاص بنا قليلا. الإيجاز ملفت للنظر: فقط 6 أسطر فعالة!

بعد استدعاء وحدة **cherrypy**، قمنا بتعريف صنف جديد **MonSiteWeb**(). الكائنات المنتجة باستخدام هذا الصنف تكون معالجات الطلبات(ألاستعلامات). يتم استدعاء أساليبها من خلال Cherrypy الداخلية، التي تحول عنوان URL المطلوب من قبل المتصفح، نطلب الأسلوب مع الاسم المعادل (سوف نوضح هذه الآلية على نحة أفضل مع المثال التالي). إذا كان عنوان URL المستلم لا يحتوي على أي اسم صفحة، كما في حالتنا، سيتم البحث عن اسم الفهرس بشكل افتراضي، بعد اتفاقية راسخة على شبكة الإنترنت. ولهذا السبب فإن اسمه أسلوب فريد من نوعه، الذي يتنظر الطلبات التي تعالج جذر الموقع .

- السطر 5 : أساليب هذا الصنف ستقوم بمعالجة الطلبات من المتصفح، وإرجاع رد كسلسلة نصية تحتوي على نص مكتوب بلغة HTML. لهذا التمرين الأول، قمنا بتبسيط كود HTML المنتج إلى أقصى حد، تم تلخيص ذلك في رسالة صغيرة بين علامات العنوان (`<h1>` و`</h1>`). بعبارة أدق، ينبغي أن تدرج كل شيء بين `<html></html>` و`<body></body>` من أجل تحقيق تخطيط صحيح. ولكن بما أنه يعمل، سوف ننتظر قليلا بعد قبل أن نريك طرق جيدة.
- السطر 6 : الأساليب التي ستقوم بمعالجة طلبات HTTP وتقوم بإرجاع صفحة ويب، يجب أن تكون "منشورة" باستخدام سمة **exposed** التي تحتوي على قيمة "صحيح". يجب أن نأمن سلامة تنفيذ CherryPy، والذي يتم افتراضيا جميع الأساليب التي كتبتها سوق يتم حمايتها في مواجهة من يحاول الوصول إليها. الأساليب الوحيدة المتاحة هي التي تم وضعها للاستخدامات العامة.

94إذا اخترت رقم المنفذ الافتراضي (80) في ملف التكوين. يجدر التذكير في العناوين أن هذا هو منفذ الذي يتم استخدامه بشكل افتراضي في معظم المتصفحات . يمكنك في هذه الحالة الاتصال بموقعك الجديد ببساطة عن طريق : *http://localhost* .

• السطر 9 : الدالة **quickstart()** لوحدة cherrypy تقوم بتشغيل الخادم الفعلي. يجب أن يتم توفير برامتر مرجع كائن معالج الطلبات الذي سيكون جذر الموقع، ومرجع الملف التكوين العام .

إضافة صفحة ثانية

نفس كائن المعالج يمكن بالطبع أن يأخذ عدة صفحات :

```
import cherrypy

class MonSiteWeb(object):

    def index(self):
        # إرجاع صفحة HTML تحتوي على رابط إلى صفحة أخرى
        # (سيتم إنشاؤها من أسلوب آخر من نفس الكائن) :
        return '''
            <h2>Veuillez <a href="unMessage">cliquer ici</a>
            pour accéder à une information d'importance cruciale.</h2>
        '''
    index.exposed = True

    def unMessage(self):
        return "<h1>La programmation, c'est génial !</h1>"
    unMessage.exposed = True

cherrypy.quickstart(MonSiteWeb(), config ="tutoriel.conf")
```

هذا السكريبت يشتق من سابقه. الصفحة تقوم بإرجاع بواسطة الأسلوب **index()** التي تحتوي هذه المرة على علامة-رابط **<a href="unMessage">** برامتر URL لصفحة أخرى. إذا كان هذا URL اسمه بسيط، من المفترض أن تكون الصفحة موجودة في الدليل الجذر للموقع. في منطق تحويل ال URL التي يتم استخدامها من قبل Cherrypy، فإنها تقوم باستدعاء أسلوب كائن الجذر مع اسم معادل. في مثالنا، صفحة المرجع سيتم إنتاجها من قبل الأسلوب **unMessage()**.

عرض ومعالجة نموذج

و سوف تصبح الأمور مثيرة للاهتمام أكثر مع السكريبت التالي :

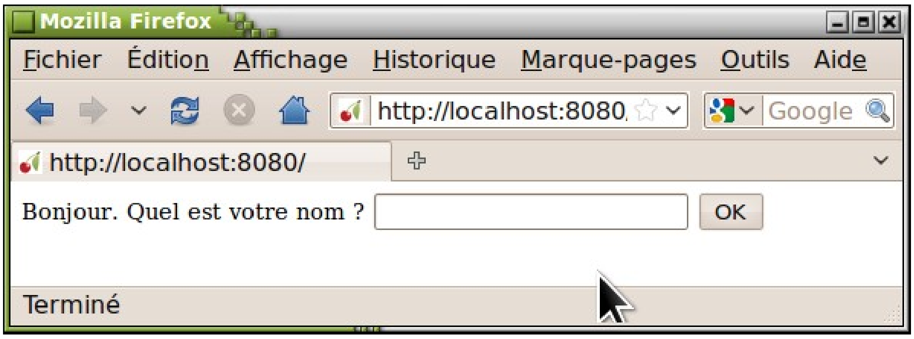
```
import cherrypy

class Bienvenue(object):
    def index(self):
        # نموذج طلب اسم من المستخدم :
        return '''
            <form action="salutations" method="GET">
            Bonjour. Quel est votre nom ?
            <input type="text" name="nom" />
            <input type="submit" value="OK" />
            </form>
        '''
    index.exposed = True

    def salutations(self, nom =None):
        if nom:            # صفحة الرئيسية للمستخدم :
```

```
17#             return "Bonjour, {0}, comment allez-vous ?".format(nom)
18#         else:                  # لم يتم كتابة أي اسم :
19#             return 'Veuillez svp fournir votre nom <a href="/">ici</a>.'
20#     salutations.exposed = True
21#
22# cherrypy.quickstart(Bienvenue(), config ="tutoriel.conf")
```

الأسلوب **index()** لكائن الجذر الخاص بنا أصبح يعرض هذه المرة للمستخدم صفحة ويب تحتوي على نموذج : كود HTML تم وضعه داخل علامات `<form>` و`</form>` والذي قد يحتوي على مجموعة من الويدجات المختلفة، والتي يمكن للمستخدم التعامل معها ، يمكنها أن تقوم بترميز المعلومات وأن تنفذ تفعيل نشاط معين : حقول الإدخال وخانات وأزرار راديو وعلب قوائم .. والخ. لهذا المثال الأول، حقل إدخال وزر كافي :



- السطر 7 : العلامة `<form>` يجب أن تحتوي على إثنين من التفاصيل الأساسية : العمل الذي ستقوم به عند إرسال النموذج (إنها في الواقع توفير URL لمصدر ويب قادر على تلقي استعلام مع البرامترات)، والأسلوب (`GET` أو `POST`) تستخدم لتمرير هذه البرامترات .

> الفرق بين *GET* و *POST* هو كيفية ربط البرامترات للاستعلام. في الرأس (*GET*). أو في ملحق (*POST*). لـ ***Cherrypy*** هذا التمييز غير مهم. يمكنك استخدام أي واحدة .

- السطر 9 يحتوي على علامات HTML التي تعرف حقل الإدخال (العلامة `<input type="text" name="nom" />`). سمة `name` تسمح بربط ملصق بسلسلة نصية التي سوف تقوم بترميزها من قبل المستخدم. عندما يقوم المتصفح بتمرير استعلام HTTP للخادم، وهذه سوف تحتوي على برامتر موصوف بشكل جيد. كما سبق وأوضحنا أعلاه، يقوم Cherrypy بتحويل هذا الاستعلام باستدعاء أسلوب كلاسيكي، والتي ترتبط مع وصف البرامتر، كالعادة في بيثون.
- السطر 10 يعرف ودجة من نوع "زر إرسال" (العلامة `<input type="submit">`). النص الذي سيتم عرضه على الزر عن طريق سمة `value`.
- الأسطر من 15 إلى 20 يتم تعريف الأسلوب الذي سيقبل الاستعلام، عندما يتم إرسال النموذج إلى الخادم. برامتر `nom` سوف تتلقى البرامتر المناسب، التي يعرفها من اسمها(هشام استبدل الوصف بالاسم) نفسه. كالعادة في بيثون، يمكنك

تحديد قيم افتراضية لكل برامتر (إذا ترك حقل النموذج فارغ من قبل المستخدم، لا يتم تمرير البرامتر). في مثالنا، البرامتر **nom** يحتوي افتراضيا كائن فارغ : سيكون من السهل جدا التأكد برمجيا مما إذا قام المستخدم فعلا بإدخال اسم أو لا .

عمل هذه الآليات هي طبيعة جدا وبسيطة جدا : يتم تحويل ال URL الذي تم أستدعاؤه من خلال Cherrypy باستدعاء أساليب تحمل نفس الاسم، والتي تمرر البرامترات بطريقة تقليدية .

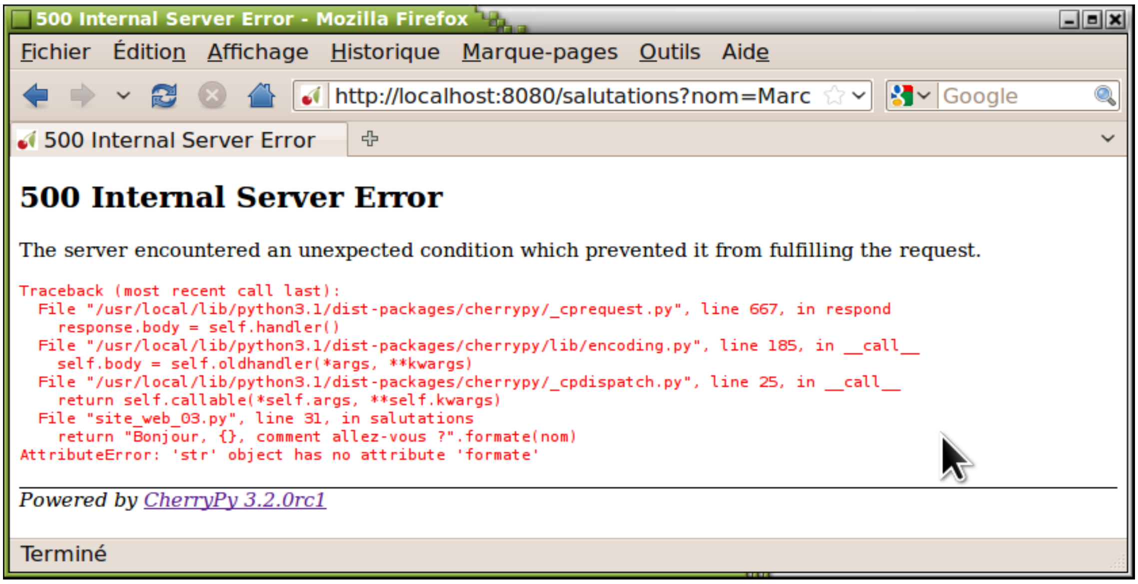
تحليل الإتصالات والأخطاء

عند تجربة السكريبتات التي تم شرحها حتى هنا، ستلاحظ أن رسائل مختلفة تظهر في نافذة الطرفية أين بدأ التنفيذ. هذه الرسائل تطلعكم (جزئيا) على الحوار الذي جرى بين الخادم والعملاء. يمكنك تأسيس اتصال مع خادمك من أجهزة أخرى (إذا كان خادمك متصل بشبكة، بطبيعة الحال) :

```
[12/Jan/2010:14:43:27] ENGINE Started monitor thread '_TimeoutMonitor'.
[12/Jan/2010:14:43:27] ENGINE Started monitor thread 'Autoreloader'.
[12/Jan/2010:14:43:27] ENGINE Serving on 127.0.0.1:8080
[12/Jan/2010:14:43:27] ENGINE Bus STARTED
127.0.0.1 - - [12/Jan/2010:14:43:31] "GET / HTTP/1.1" 200 215 "" "Mozilla/5.0
(X11; U; Linux i686; fr; rv:1.9.1.6) Gecko/20091215 Ubuntu/9.10 (karmic)
 Firefox/3.5.6"
127.0.0.1 - - [12/Jan/2010:14:44:07] "GET /salutations?nom=Juliette HTTP/1.1"
 200 39 "http://localhost:8080/" "Mozilla/5.0 (X11; U; Linux i686; fr;
 rv:1.9.1.6) Gecko/20091215 Ubuntu/9.10 (karmic) Firefox/3.5.6"
```

في نافذة الطرفية حين تجد رسائل الخطأ (على سبيل المثال، أخطاء في تركيب الجملة) التي تتصل ببرنامج في نهاية المطاف قبل بدئ تشغيل الخادم. فإذا تم الكشف عن خطأ أثناء العمل (خطأ في أسلوب معالج الاستعلامات)، رسالة الخطأ تظهر في نافذة المتصفح، والخادم يواصل عمله. الشكل في الصفحة 334 تظهر على سبيل المثال رسالة التي حصلنا عليها من خلال إضافة خطأ لاسم أسلوب **format**()، في السطر 17 من السكريبت الخاص بنا (formate(nom) بدلًا من format(nom)).

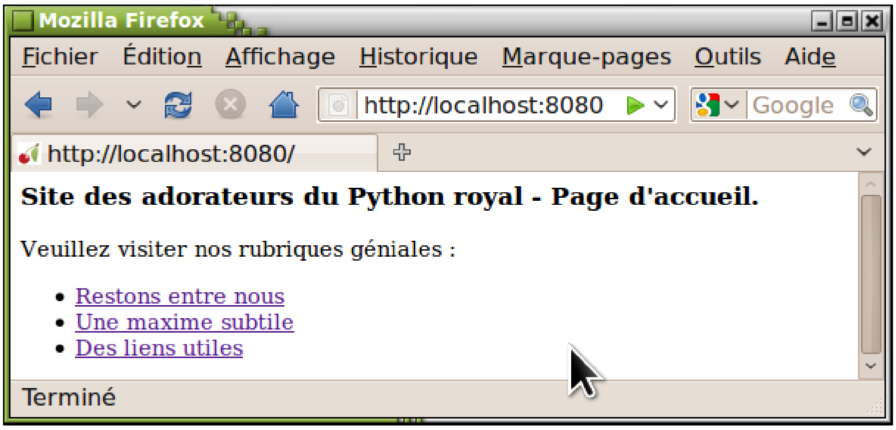
يمكنك التحقق من أن الخادم يعمل دائما، من خلال العودة إلى الصفحة السابقة وإدخال هذه المرة اسم فارغ. هذا الخيار لا يمنع عندما يتم الكشف عن خطأ في غاية الأهمية لخادم الويب، لأن هذا سوف يسمح بالاستمرار في تلبية معظم طلبات التي يتلقاها، حتى لو كان ينبغي أن يرفض بعضها لأن لا يزال هنالك بعض مشاكل الصغير في البرنامج. وهذا قد يبدو من الوهلة الأول، لأنه حتى الآن ستجد دئما أن خطأ في وقت التنفيذ يسبب في إغلاق البرنامج .



هذا لا يحدث هذه المرة، لأن عند بدأ تشغيل تطبيقنا، Cherrypy يطلق عدة أسطر تنفيذ البرنامج والتي نسميها threads (مواضيع)، وواحد فقط تم إيقافه بالخطأ. وسيتم شرح المزيد من threads- المواضيع بالتفصيل في الفصل 19 (صفحة384).

هيكل موقع متعدد الصفحات

سوف نرى الآن كيفية هيكلة موقعنا، وإنشاء تسلسل هرمي للأصناف بطريقة مماثلة لتلك بين الدلائل والدلائل الفرعية في نظام الملفات .



في السكريبت أدناه، سوف نولي اهتماما خاصا بتعريف العلامات-الوصلات <a href=...> :

```
import cherrypy

class HomePage(object):
    def __init__(self):
```

```
        # كائنات معالجة للطلبات(إستعلامات) يمكنها أن تقوم بتمثيل نفسها كمعالجات أخرى "عبيدة", وهكذا :
        self.maxime = MaximeDuJour()
        self.liens = PageDeLiens()
        # يمكن بالطبع أن تقوم بتمثيل كائنات معالجة الإستعلامات في أي مستوى من البرنامج .

    def index(self):
        return '''
            <h3>Site des adorateurs du Python royal - Page d'accueil.</h3>
            <p>Veuillez visiter nos rubriques géniales :</p>
            <ul>
                <li><a href="/entreNous">Restons entre nous</a></li>
                <li><a href="/maxime/">Une maxime subtile</a></li>
                <li><a href="/liens/utiles">Des liens utiles</a></li>
            </ul>
        '''
    index.exposed = True

    def entreNous(self):
        return '''
            Cette page est produite à la racine du site.<br />
            [<a href="/">Retour</a>]
        '''
    entreNous.exposed =True

class MaximeDuJour(object):
    def index(self):
        return '''
            <h3>Il existe 10 sortes de gens : ceux qui comprennent
            le binaire, et les autres !</h3>
            <p>[<a href="../">Retour</a>]</p>
        '''
    index.exposed = True

class PageDeLiens(object):
    def __init__(self):
        self.extra = LiensSupplementaires()

    def index(self):
        return '''
        <p>Page racine des liens (sans utilité réelle).</p>
        <p>En fait, les liens <a href="utiles">sont plutôt ici</a></p>
        '''
    index.exposed = True

    def utiles(self):
        # لاحظ كيف تم تعريف الارتباط إلى الصفحات الأخرى :
        # on peut procéder de manière ABSOLUE ou RELATIVE.
        return '''
            <p>Quelques liens utiles :</p>
            <ul>
                <li><a href="http://www.cherrypy.org">Site de CherryPy</a></li>
                <li><a href="http://www.python.org">Site de Python</a></li>
            </ul>
            <p>D'autres liens utiles vous sont proposés
            <a href="./extra/"> ici </a>.</p>
            <p>[<a href="../">Retour</a>]</p>
        '''
    utiles.exposed = True

class LiensSupplementaires(object):
```

```
65#     def index(self):
66#         # لاحظ الارتباط النسبي للعودة إلى الصفحة الرئيسية :
67#         return '''
68#             <p>Encore quelques autres liens utiles :</p>
69#             <ul>
70#                 <li><a href="http://pythomium.net">Le site de l'auteur</a></li>
71#                 <li><a href="http://ubuntu-fr.org">Ubuntu : le must</a></li>
72#             </ul>
73#             <p>[<a href="../">Retour à la page racine des liens</a>]</p>
74#         '''
75#     index.exposed = True
76#
77# racine = HomePage()
78# cherrypy.quickstart(racine, config ="tutoriel.conf")
```

الأسطر من 4 إلى 10 : أسلوب منشئ الكائنات الجذر هو المكان المثالي لتمثيل كائنات أخرى "عبيد". فإننا نصل إلى أساليب معالجة الاستعلامات وهذا تماما كما يمكننا الوصول إلى الدلائل الفرعية من الدليل الجذر (انظر أدناه).

الأسطر من 12 إلى 22 : إن صفحة الرئيسية توفر روابط للصفحات الأخرى للموقع. ستلاحظ أن بيئة الجمل المستخدمة في علامات -الوصلات، تستخدم هنا لتعريف مسار كامل :

- أساليب الكائن الجذر يتم عمل مرجع لها عن طريق الرمز / متبوع باسم واحد. الرمز / يشير أن "مسار" جزء من الجذر الموقع. على سبيل المثال: `/entreNous`.
- إن أساليب الجذر الكائنات العبيد يتم عمل مرجع لهم باستخدام رمز / بسيط ثم اسم هذه الكائنات الأخرى. على سبيل المثال: `/maxime/`
- أساليب الكائنات العبيد يتم عمل مرجع لها باستخدام اسم مدرج في مسار الكامل : على سبيل المثال: `/liens/utiles`

الأسطر 36 و 62 و 75 : لإيجاد جذر المستوى السابق، استخدمنا هذه المرة مسار نسبي، مع نفس تكوين الجملة التي استخدمت للعودة إلى الدليل السابق في نظام الملفات (نقطتين).

الأسطر 41 و 42 : عليك أن تعرف أننا ثبتنا تسلسلا هرميا في شكل شجرة ملفات، وقمنا بتمثيل كائنات "العبيد" عن بعضها البعض. بعد هذا المنطق، ينبغي أن يكون المسار الكامل مؤديا إلى الأسلوب **index()** لهذا الصنف أن يكون `/liens/extra/index`.

دعم الجلسات

عند تطوير موقع إلكتروني تفاعلي، نحن نأمل أن الشخص يقوم بزيارات متكررة للموقع، ويمكن تحديد وتوفير عدد من المعلومات في زياراته في كامل الصفحات المختلفة (على سبيل المثال، ملء سلة من خلال التشاور من موقع تجاري)، وكل هذه المعلومات سوف يتم تخزينها في مكان ما حتى ينهي زيارته. بالطبع، يجب علينا أن نربطه بكل عميل متصل بشكل مستقل، لأننا لا يمكن أن ننسى أن الغرض الموقع على شبكة الإنترنت هو استخدامه جنبا إلى جنب مع مجموعة متنوعة من الأشخاص.

سيكون من الممكن نقل هذه المعلومات من صفحة إلى صفحة أخرى، وذلك باستخدام حقول النموذج المخفي (العلامة < INPUT TYPE="hidden">). لكن هذا سيكون معقدا ومرهقا. ولذلك سيكون من الأفضل تجهيز الخادم مع آلية خاصة، لتعيين جلسة خاصة لكل عميل، والتي سوف تقوم بتخزين جميع المعلومات الخاصة بهذا العميل. Cherrypy يحقق هذا الهدف من خلال ملفات تعريف الارتباط (كوكيز).

عندما يدخل زائر جديد لهذا الموقع، سيقوم الخادم بإنشاء ملف تعريف الارتباط (و هذا يعني حزمة صغيرة من المعلومات تحتوي على معرف جلسة فريدة من نوعها لسلسلة عشوائية من وحدات البايت) وتقوم بإرجاعها لمتصفح الويب، الذي يقوم بحفظها. في اتصال مع ملف التعريف الذي تم عمله، سيقوم الخادم بالاحتفاظ لبعض الوقت كائن-جلسة الذي سيقوم بتخزين جميع المعلومات الخاصة بالزائر.

في اتصال مع الكوكيز(ملف تعريف الارتباط) الذي تم صنعه، سيقوم الخادم بالاحتفاظ لبعض الوقت كائن-جلسة الذي سيتم حفظ جميع معلومات الخاصة بالزائر. عند فتح صفحات الموقع الأخرى، يقوم المتصفح بإعادة إرسال كل مرة محتوى كوكيز الخادم، وهذا يسمح بتحديد واسترداد كائن-الجلسة. لا يزال كان-الجلسة متاح في جميع الصفحات التي يزورها الزائر : هو كائن بيثون عادي، يتم من خلاله تخزين أي عدد من المعلومات في شكل سمات.

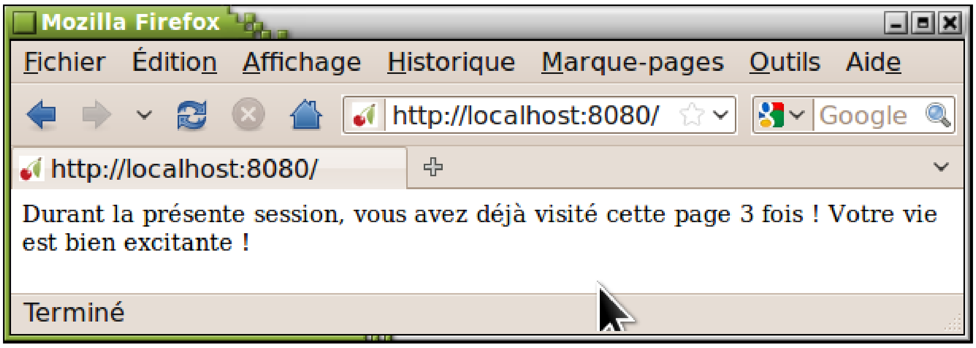
من حيث البرمجة، هكذا تسير الأمور :

```
import cherrypy

class CompteurAcces(object):
    def index(self):
        # مثال بسيط : تزايد العداد.
        # نبدأ بإسترجاع إجمالي العد الحالي :
        count = cherrypy.session.get('count', 0)
        # ... زيادة ما يلي :
        count += 1
        # ... تخزين القيمة الجديدة في قاموس الجلسة :
        cherrypy.session['count'] = count
        # ... ونقوم بعرض العد الحالي :
        return '''
            Durant la présente session, vous avez déjà visité
            cette page {0} fois ! Votre vie est bien excitante !
        '''.format(count)
    index.exposed = True

cherrypy.quickstart(CompteurAcces(), config='tutoriel.conf')
```

يمكنك ببساطة إعادة طلب الصفحة التي ينتجها هذا السكريبت مرارا وتكرارا ستلاحظ أنه في كل مرة يزداد فيها عداد الزيارة مرة، كما هو مبين في الشكل أدناه.



يجب أن يكون السكريبت يفسر نفسه بنفسه. ويجدر الإشارة إلى أن وحدة **cherrypy** تم تكييفها لكائن **session** الذي يتصرف (على ما يبدو) كقاموس كلاسيكي. يمكننا أن نضيف مفاتيح لإدارتها، ونربطها بمفاتيح أي قيم.

يمكنك ببساطة إعادة طلب الصفحة التي ينتجها هذا السكريبت مرارا وتكرارا ستلاحظ أنه في كل مرة يزداد فيها عداد الزيارة مرة، كما هو مبين في الشكل أدناه.

في السطر 7 من مثالنا، نحن استخدمنا الأسلوب **get()** للقواميس، لإيجاد القيمة المرتبطة بمفتاح **count** (أو صفر، إذا كان المفتاح غير موجود). في السطر 11 سنقوم بتسجيل هذه القيمة، المتزايدة، في نفس القاموس. وهكذا يمكننا أن نرى مرة أخرى أن Cherrypy يوفر لنا بيئة برمجية مألوفة لبيثون العادية.

لاحظ دائما أن الكائن **session**ـ الذي يتصرف لنا كقاموس بسيط، هو في الواقع آلية داخلية معقدة، لأننا نخدم تلقائيا المعلومات التي تتوافق مع عميل معين من موقعنا، الذي حدد عن طريق الكوكيز الجلسة. لترى هذا، جرب الوصول إلى خادمك من خلال متصفحين مختلفين[95] (على سبيل المثال،فايرفوكس وأوبرا) : سوف تجد أن عدد الزيارات هو في الواقع مختلف لكل واحدة منهم .

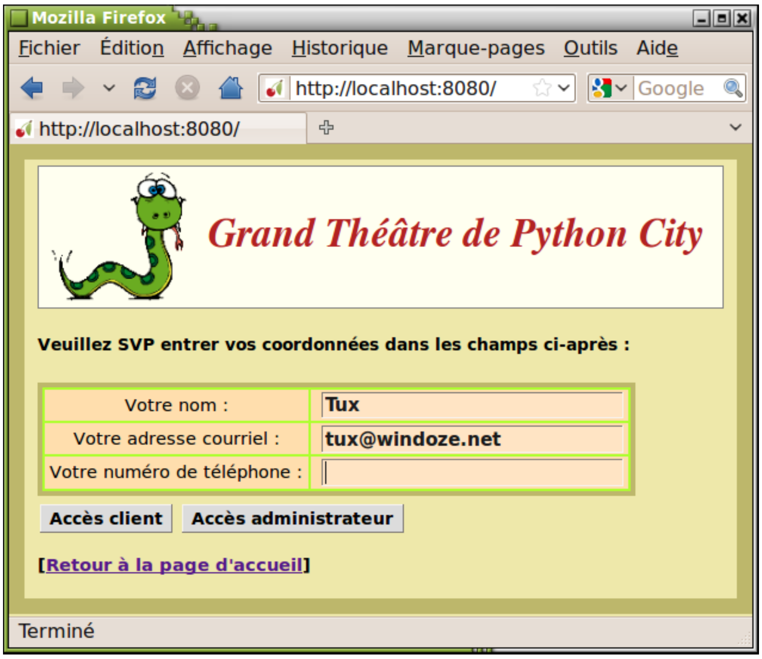
عمل موقع تفاعلي ملموس على شبكة الإنترنت

مع كل ما تعلمناه حتى الآن، يجب أن نكون قادرين على النظر إلى مشروع أكثر أهمية. هذا ما سنقدمه أدناه، تفاصيل تطوير موقع على شبكة الإنترنت يمكن استخدامه نوعا ما لحجز مقعد في المسرح[96] :

---

95 انتباه : لا يمكن تمييز الجلسات سوى من بداية الآليات المختلفة (إستخدام أي متصفح). أو المتصفحات المختلفة التي تعمل على نفس الجهاز . فإذا قمت بتشغيل مثيلين من المتصفح نفسه على نفس الجهاز سوف تستخدم ملفات تعريف الارتباط نفسها. و هذا يعني أن الخادم لا يمكنه التمييز بين الطلبات الواردة من أي واحدة . و بعبارة أخرى فإن هذين المثيلين من المتصفح نفسه يشاركون نفس الجلسة .

96 رسم الثعبان هو شعار لبرنامج حر WebCamSpy (و هذا البرنامج كتب ببيثون) .
Voir : http://webcamspy.sourceforge.net/



يمكن لـ "مدراء" الموقع إضافة مشاهدين إلى القائمة وعرض الحجوزات. ويمكن "للعملاء" التسجيل وحجز أماكن لمشاهد أو أكثر، ويمكنه مشاهدة الأماكن الذي اشتراها.

في الحقيقة يوجد دائما، تمرين، وظائف هذا التطبيق الصغير ربما قد تكون غير مكتملة. فهي تفتقر إلى تحكم دخول المدراء (على سبيل المثال،عن طريق نظام تسجيل دخول)، والقدرة على حذف أو تعديل المشاهدين الحاليين والحجوزات، ومعالجة التواريخ وعناوين البريد الالكتروني وأرقام الهواتف (هنا سلاسل نصية بسيطة)، إلخ.

التطبيق يتضمن قاعدة بيانات صغيرة علائقية تضم ثلاثة جداول، وتربط بواسطة اتصالين من نوع "واحد إلى العديد". تم صنع صفحات الويب بتخطيط مشترك، والديكورها تم استخدام CSS. كودات تطبيق بيثون وكودات HTML "الرئيسة" منفصلة في ملفين منفصلين.

لقد قمنا بدمج هذا المثال بأقصى حد من المفاهيم المفيدة، لكن تم عمدا ترك كود التحكم الذي من شأنه أن تكون له حاجة حقيقية في التطبيق للتحقق من مدخلات المستخدم، والكشف عن أخطاء الاتصال مع قاعدة البيانات ... إلخ ، حتى لا تشوه المظاهر .

بغض النظر عن قاعدة البيانات (و التي سوف تعتمد على SQLite)، ينقسم الجزء الأساسي من التطبيق إلى 3 ملفات منفصلة، وذلك لفصل معالجة البيانات، وتجهيزه المستخدم وتزيين الوقع :



- معالج البيانات يتم توفيره من قبل سكريبت بيثون الذي سنصفه بعد قليل. ويتم تضمينه في ملف واحد spectacles.py لكن نحن نقدم لكم عدة قطع لتسهيل الشرح وعرض النص.

  هذا السكريبت يستخدم آلية الجلسات لتذكر تفاصيل المستخدم خلال زيارته .

- إن تقديم البيانات تضمنه مجموعة من صفحات الويب، بما في ذلك كود HTML الذي يتم ترجمة الجزء الأكبر منه في ملف نصي منفصل spectacles.htm. حسب البرنامج، نقوم باستخراج سلاسل نصية من هذا الملف سيتم تنسيقها عن طريق إدخال قيم متغيرات، وذلك باستخدام التقنية الموضحة في الفصل 10. وسوف تكون صفحات المحتوى ثابتة لا تشوش السكريبت نفسه. سوف نصنع محتويات هذا الملف في الصفحة 350.

- تزيين الصفحات سيتم عن طريق استخدام ورقة الأنماط CSS، والتي سوف تكون في ملف منفصل spectacles.css. نحن لن نشرح في هذه الصفحات كود CSS المستخدم، لأن هذا خارج نطاق دورة برمجة بيثون. لقد قمنا بإدراج أسماء وثائق يمكن تحميلها من موقع هذا الكتاب .

السكريبت

يبدأ سكريبت **spectacles.py** بتعريف صنف **Glob()** الذي سيستخدم كحاوي للمتغيرات التي نريد معالجتها كمتغيرات عامة. ويتضمن وصف الجداول في قاعدة البيانات في قاموس، وذلك باستخدام تقنية مشابهة لما تم شرحه في الفصل السابق :




```
1#  import os, cherrypy, sqlite3
2#
3#  class Glob(object):
4#      # Données à caractère global pour l'application
5#      patronsHTML ="spectacles.htm"          # الملف الذي يحتوي على "علامات" HTML
6#      html ={}                # Les patrons seront chargés dans ce dictionnaire
7#      # سيتم تحميل العلامات في هذا قاموس هيكل قاعدة البيانات. قاموس الجداول والحقول :
8#      dbName = "spectacles.sq3"              # اسم قاعدة البيانات
9#      tables ={"spectacles":(("ref_spt","k"), ("titre","s"), ("date","t"),
10#                             ("prix_pl","r"), ("vendues","i")),
11#               "reservations":(("ref_res","k"), ("ref_spt","i"), ("ref_cli","i"),
12#                               ("place","i")),
13#               "clients":(("ref_cli","k"), ("nom","s"), ("e_mail","s"),
14#                          ("tel", "i")) }
```

تليها تعاريف ثلاثة دالات. الأول (من السطر 16 إلى 33) لن تستخدم سوى مرة واحدة عند بدء التشغيل. دورها هو قراءة ملف النصي **spectacles.htm** لاستخراج كودات HTML التي سوف تستخدم لتنسيق صفحات الويب. فإذا فحصت هيكل هذا الملف (لقد قمنا بتكرار محتوى الملف في الصفحات من 350-351)، سوف ترى أنه يحتوي على سلسلة من الأقسام، واضحة من كل واحد من التكرارين : علامة الفتح (شكلت نفسها من نجمتين وخطافين) وسطر أخير يتكون من ما لا يقل عن خمسة رموز #. ويمكن استخراج كل قسم عن حدة وتخزينها في قاموس عام (**glob.html**). سيتم العثور على مفاتيح هذا القاموس وقيم الأقسام المقابلة، كل واحد منها يحتوي على صفحة من كود HTML، مع علامات التحويل {0}، {1}، {2}، إلخ. والذي يمكن استبدالها بقيم متغيرات. ويجب أن يؤخذ في الاعتبار ترميز هذا الملف النصي، بالطبع (السطر 18) .

```
def chargerPatronsHTML():
    # تحميل جميع "علامات" صفحات ال HTML في قاموس
    # (يتم تحديد الترميز, في حال إذا كان يختلف عن الافتراضي) :
    fi =open(Glob.patronsHTML, "r", encoding ="Utf8")
    try:        # لضمان أن يتم إغلاق الملف دائما
        for ligne in fi:
            if ligne[:2] =="[*":                 # ==> العثور على التسمية
                label =ligne[2:]                 # مسح [*
                label =label[:-1].strip()        # suppression LF et esp évent.
                label =label[:-2]                # مسح *]
                txt =""
            else:
                if ligne[:5] =="#####":
                    Glob.html[label] =txt
                else:
                    txt += ligne
    finally:
        fi.close()          # سيتم إغلاق الملف في جميع الحالات

```

الدالة الثانية، على الرغم من أنها بسيطة، فهي تؤدي عملا رائعا : بل في الواقع من شأنه أن يسمح لنا بتقديم جميع الصفحات المتشابهة ، بإدراجها في نمط شائع. هذا النمط، مثل جميع الأخرين، يأتي من ملف spectacles.htm لكن الدالة السابقة قد قدمته لنا في قاموس **Glob.html**، تحت اسم "miseEnPage" :

```
def mep(page):
    # توليد دالة "التخطيط" لل HTML : تقوم بإرجاع <الصفحة> الممررة، التي وضع بها برامتر في رأس وتذييل الصفحة الملائمة
    return Glob.html["miseEnPage"].format(page)

```

الدالة الثالثة تصنع قطعة قطعة سلسلة نصية تحتوي على كود HTML الازم لوصف الجدول. هذا الجدول سيتم ملئه تلقائيا مع قائمة البرامج المدرجة حاليا في قاعدة البيانات. نرى هنا بشكل واضح جدا مساهمة البرمجة الأساسية لتحقيق موقع ديناميكي، وهذا يعني موقعا يتم تحديث صفحاته باستمرار استنادا إلى البيانات التي يقدمها الزوار بأنفسهم. أو وفقا لمختلف الأحداث :

```
def listeSpectacles():
    # إنشاء قائمة من العروض المتاحة، في جدول HTML.
    req ="SELECT ref_spt, titre, date, prix_pl, vendues FROM spectacles"
    res =BD.executerReq(req)            # ==> res sera une liste de tuples
    tabl ='<table border="1" cellpadding="5">\n'
    tabs =""
    for n in range(5):
        # ملاحظة : لتظهر كسلسلة منسقة، يجب أن تقوم بمضاعفة الأقواس
        tabs += "<td>{{{0}}}</td>".format(n)
    ligneTableau ="<tr>" +tabs +"</tr>\n"
    # السطر الأول من الجدول تحتوي على رؤوس الأعمدة :
    tabl += ligneTableau.\
        format("Réf.", "Titre", "Date", "Prix des places", "Vendues")
    # الأسطر التالية : يتم استخراج محتواها من BD :
    for ref, tit, dat, pri, ven in res:
        tabl += ligneTableau.format(ref, tit, dat, pri, ven)
    return tabl +"</table>"

```

الأسطر 42 و 43 ستستعلم عن قاعدة البيانات لاستخراج المعلومات حول العروض المتاح من خلال صنف-الواجهة الذي تم وصفه في وقت لاحق. عند بداية السكريبت، سوف يتم إنشاء مثيل يسمى **BD** كائن من هذا الصنف، يقوم الأسلوب **executerReq()** بإرجاع نتيجة استعلام SQL في السطر 42 في شكل سلسلة ومصفوفات مغلقة.

الأسطر التالية تظهر لك كيفية بناء جدول HTML من خلال البرمجة، والاستفادة من تقنيات السلاسل التنسيق التي تم وصفها في الفصل 10.

الأسطر من 45 إلى 50 تقوم ببناء أولا سلسلة التنسيق :

`"<tr><td>{0}</td><td>{1}</td><td>{2}</td><td>{3}</td><td>{4}</td></tr>"`

لتكون بمثابة قالب رئيس لتوليد كود HTML يصف أسطر الجدول. لاحظ الحاجة إلى مضاعفة الرموز { و } بحيث يتم إدراجها على هذا النحو في السلسلة.

الأسطر التالية تقوم ببناء كود HTML بنفسها، وتستكمل السلسلة التي تم بدأها في السطر 44، مع وصف كامل لأسطر الجدول. علامات التنسيق للرئيس يتم استبدالها واحدة تلو الأخرى بالمعلومات التي تم استخراجها من قاعدة البيانات (تدوير قائمة المصفوفات المغلقة، الأسطر 55 و 56)، ويتم إنهاء السلسلة بإرجاع للبرنامج الذي تم استدعاؤه في السطر 57 :

إن **GestionBD()** تضمن التواصل مع قاعدة البيانات. والتي هي نسخة مبسطة من صنف بنفس الاسم تم وصفه في نهاية الفصل السابق. تم تحسينه بالتأكيد[97]. وقاعدة البيانات تحتوي بنفسها تماما مثل الملف spectacles.sq3. فإذا قمت بحذف هذا الملف، سيكون من الممكن إعادة صنعه تلقائيا من خلال الأسلوب **creaTables()** :

```
57# class GestionBD(object):
58#     # التنفيذ والتواصل مع قاعدة البيانات SQLite.
59#
60#     def __init__(self, dbName):                    # انظر إلى ملاحظات
61#         self.dbName =dbName                        # الكتاب حول الخيوط
62#
63#     def executerReq(self, req, param =()):
64#         # تشغيل الاستعلام <req>, مع إرسال النتيجة المحتملة
65#         connex =sqlite3.connect(self.dbName)     # إنشاء المؤشر
66#         cursor =connex.cursor()                  # تنفيذ استعلام SQL
67#         cursor.execute(req, param)
68#         res =None
69#         if "SELECT" in req.upper():
```

[97] سترى على وجه الخصوص أننا أعدنا صنع كائن اتصال جديد عند كل استعلام, و هذا ليس بالأمر السعيد لكن فعلنا هذا لأن SQLite لا تسمح باستخدام ضمن خيط خاص, و كائن إتصال الذي تم إنشاؤه في خيط اخر . (و الخيوط هي أسطر برمجية يتم تنفيذه بالتوازي في نفس البرنامج . و تطبيق ويب مثل Cherrypy هو إحداها, و ذلك من أجل المعالجة في نفس الوقت تقريبا استعلامات المستخدمين المختلفة . و سيتم شرح الخيوط في الفصل 19) . إذا كنت تريد صنع كائن اتصال واحد لمعالجة مجموعة من الاستعلامات, و ينبغي أن : إما أن لا يستخدم البرنامج في كل شيء سوى خيط واحد, و إما إضافة الوظيفة المطلوبة لتعريف الخيوط الحالية, و إما استخدام SGBDR أخر غير SQLite (على سبيل المثال, PostgreSQL) . و كل هذا سوف يكون ممكنا, و لكن يتطلب الكثير من الشرح و هذا سيكون خارج نطاق هذا الكتاب .

```
            res =cursor.fetchall()           # <res> = قائمة مصفوفات مغلقة
        connex.commit()                      # تسجيل منتظم
        cursor.close()
        connex.close()
        return res                   # سوف يرجع None أو قائمة من المصفوفات
المغلقة

    def creaTables(self, dicTables):
        # إنشاء جداول من قاعدة البيانات إذا لم تكن موجودة بالفعل
        for table in dicTables:              # تدوير مفاتيح القاموس
            req = "CREATE TABLE {0} (".format(table)
            pk =""
            for descr in dicTables[table]:
                nomChamp = descr[0]          # libellé du champ à créer
                tch = descr[1]               # type de champ à créer
                if tch =="i":
                    typeChamp ="INTEGER"
                elif tch =="k":
                    # حقل "مفتاح أساسي" (عدد صحيح متزايد تلقائيا
                    typeChamp ="INTEGER PRIMARY KEY AUTOINCREMENT"
                    pk = nomChamp
                elif tch =="r":
                    typeChamp ="REAL"
                else:                        # للتبسيط, سوف نعتبر جميع الأنواع الاخرى
كنصوص
                    typeChamp ="TEXT"
                req += "{0} {1}, ".format(nomChamp, typeChamp)
            req = req[:-2] + ")"
            try:
                self.executerReq(req)
            except:
                pass                         # الجدول على الأرجح موجود بالفعل
```

يتم اعتماد الوظيفة الرئيسية للموقع بواسطة الصنف التالي: **WebSpectacles**(). عند بداية السكريبت، يقوم Cherrypy بتمثيل عدة كائنات من هذا الصنف لمواضيع (threads) مختلفة من أجل صنع إستعلامات من مختلف المستخدمين بالتوازي ، هذه الآلية تعدد المهام سوف يتم شرحها في الفصل القادم، لكن يجب أن لا تقلق بشأن ذلك الآن لأن Cherrypy يوفر طريقة أكثر شفافية. لفهم ما يلي، أنت تعرف أن Cherrypy يقوم بتحويل كل واحدة من طلبات URL المطلوبة من قبل متصفح الويب الزائر باستدعاء أساليب هذا الصنف، ما قمنا بوصفه له موجز في الصفحات الأولى من هذا الفصل

```
class WebSpectacles(object):
    # صنف مولد كائنات معالجة استعلامات HTTP.

    def index(self):
        # الصفحة الرئيسية للموقع. وسوف تستخدم متغيرات الجلسة لتحديد العمليات التي تجري
بالفعل (أو لا) من قبل الزار : :
        nom =cherrypy.session.get("nom", "")
        # إرجاع صفحة HTML تتكيف مع وضع الزائر :
        if nom:
            acces =cherrypy.session["acces"]
            if acces =="Accès administrateur":
                # إرجاع صفحة HTML "ثابتة" :
                return mep(Glob.html["accesAdmin"])
```

```
113#              else:
                      # إرجاع صفحة HTML منسقة مع اسم الزائر :
115#                  return mep(Glob.html["accesClients"].format(nom))
116#          else:
117#              return mep(Glob.html["pageAccueil"])
118#      index.exposed =True
119#
```

يتم إنشاء الصفحة الأولى من قبل الأسلوب **index()**. وهذا شكل من أشكال HTML، ثم سنقوم بتحميل الكود في قاموس **Glob.html** (تحت اسم "pageAccueil") أثناء مرحلة تهيئة البرنامج. سيقوم الأسلوب **index()** بإرجاع هذا الكود (في السطر 120)، لكن فقط إذا لم يحدد زائر الموقع. أو إذا حدد، سوف تحتوي القيم في كائن-الجلسة على، cherrypy.session["nom"] و cherrypy.session["acces"]ـ سيقوم البرنامج بإرجاع الإحالات المرجعية لصفحات أخرى على الويب، الأنماط التي سوف يتم تخزينها في Glob.html (الأسطر من 111 إلى 118).

جميع الصفحات سيقوم بإرجاع "ما يرتدونه" مسبقا باستخدام الدالة **mep()**ـ التي سوف "تزين" بطريق مماثلة للمؤشرات التالية التي هي ورقة أنماط CSS. في السطر 118 سيقوم بالجمع بين اثنين من التنسيقات المتتالية، الأول لدمج كود HTML المنتج محليا (اسم تم إدخاله من قبل المستخدم) سيتم إستخراج النمط من قاموس **glob.html**، والثانية لـ"الالتفاف" على كل رئيس آخر، عبر دالة **mep()**.

الأسلوب التالي معقد قليلا. لفهم وظيفته، فمن الأفضل أن نتفحص أولا محتويات الصفحة الرئيسية التي تم إرجاعها للمستخدم عن طريق الأسلوب **index()** (في السطر 120). هذه الصفحة تحتوي على شكل HTML حتى يتسنى لنا إنتاجه فيما بعد. ويحدها نموذج مثل هذا عن طريق علامات <form> و</form> :

```
<form action="/identification" method=GET>
<h4>Veuillez SVP entrer vos coordonnées dans les champs ci-après :</h4>
<table>
<tr><td>Votre nom :</td><td><input name="nom"></td></tr>
<tr><td>Votre adresse courriel :</td><td><input name="mail"></td></tr>
<tr><td>Votre numéro de téléphone :</td><td><input name="tel"></td></tr>
</table>
<input type=submit class="button" name="acces" value="Accès client">
<input type=submit class="button" name="acces" value="Accès administrateur">
</form>
```

تستخدم سمة action في علامات <form> تشير إلى الذي سيتم استدعاؤه عندما ينقر الزائر على أحد أزرار من نوع إرسال (submit). هذا العنوان (URL) سيتم تحويله عبر Cherrypy باستدعاء أسلوب من نفس الاسم، على جذر الموقع حيث أن الاسم السابق بسيط"/". ولذلك فإن الأسلوب identification() لصنفنا الرئيسي هو الذي سيتم استدعاؤه. وإن علامات من نوع <input name= "..."> تقوم بتعريف حقل الإدخال، ولكل واحدة منها اسمها الخاص الذي تم الإشارة إليه بسمة name. وهذه التسميات التي تسمح لـ Cherrypy بتمرير قيم تم ترميزها داخل الحقول، إلى برامترات من نفس أسماء الأسلوب identification(). انظر الآن إلى هذا :

```
120#     def identification(self, acces="", nom="", mail="", tel=""):
121#         # يتم تخزين إحداثيات الزائر في متغير الجلسة :
122#         cherrypy.session["nom"] =nom
123#         cherrypy.session["mail"] =mail
124#         cherrypy.session["tel"] =tel
125#         cherrypy.session["acces"] =acces
126#         if acces =="Accès administrateur":
127#             return mep(Glob.html["accesAdmin"])
128#         else:
129#             # يستخدم لحجز مقاعد للمشاهدين التي اختارها الزائر "caddy" متغير الجلسة :
130#             cherrypy.session["caddy"] =[]          # (قائمة فارغة، في البداية)
131#             return mep(Glob.html["accesClients"].format(nom))
132#     identification.exposed =True
133#
```

البرامترات التي تم تلقيها في المتغيرات المحلية **mail**، **nom**، **acces** و **tel**. نأمل أن يتم تخزين هذه القيم الخاصة لكل مستخدم، وهو مايدفعنا إلى إسناد إلى رعاية كائن-الجلسة **cherrypy.session**ـ الذي يقدم لنا تحت الغطاء قاموس بسيط (الأسطر من 125 إلى 128 و 134).

السطر 129 : البرامتر **acces** سيقوم بتلقي قيمة المقابلة لزر الإرسال (`submit`) الذي سيستخدم من قبل الزائر، أي المسؤولين عن "دخول المديرين" أو "دخول العملاء". وهذا يسمح لنا بتوجيه الزوار إلى الصفحات التي تناسبهم .

السطر 134 : سو نقوم بحفظ الحجوزات التب طلبها الزائر في قائمة من المصفوفات المغلقة، التي سوف تملأ بطريقة خاصة. قياسا على ما يتم على المواقع التجارة الإلكترونية على شبكة الإنترنت، سوف ندعو هذه القائمة بـ "سلة - panier" أو "عربة تسوق - caddy". وأما عن حفظ هذه الحجوزات في قاعدة البيانات سيتم في وقت لاحق، في خطوة منفصلة، وفقط عندما يريد المستخدم حفظ هذه القائمة المحددة طوال فترة زيارة الموقع، ونحن قد وضعنا في متغير يسمى `"caddy"` (سوف نسمي الآن متغيرات الجلسة القيم المخزنة في كائن-الجلسة **cherrypy.session**).

تقوم صفحة الوب بإرجاع صفحة بسيطة ثابتة للمستخدم "العميل" (السطر 135)، التي توفر وصلات لصفحات أخرى. بقية السكريبت يحتوي على الأساليب المقابلة :

```
134#     def reserver(self):
135#         # تقديم نموذج الحجز إلى الزائر "العميل" :
136#         nom =cherrypy.session["nom"]              # البحث عن اسمه
137#         # قائمة المشاهدين المقترحة BD البحث في :
138#         tabl =listeSpectacles()
139#         return mep(Glob.html["reserver"].format(tabl, nom))
140#     reserver.exposed =True
141#
142#     def reservations(self, spect="", places=""):
143#         # تخزين الحجوزات المطلوبة, في متغير الجلسة :
144#         spect, places = int(spect), int(places)   # تحويل إلى أرقام
145#         caddy =cherrypy.session["caddy"]          # إستعادة الوضع الحالي
146#         caddy.append((spect, places))             # إضافة مصفوفة مغلقة إلى القائمة
147#         cherrypy.session["caddy"] =caddy          # تخزين القائمة
148#         nSp, nPl = len(caddy), 0
149#         for c in caddy:                           # مجموع الحجوزات réservations
150#             nPl += c[1]
```

```
151#          return mep(Glob.html["reservations"].format(nPl, nSp))
152#      reservations.exposed =True
153#
```

الأسطر من 138 إلى 144 : هذا الأسلوب يولد استمارة حجز في المسارح الموجودة. فهي تستخدم الدالة **listeSpectacles()**، المذكورة أعلاه، لتوليد قائمة كجدول HTML. ثم دمج ذلك في تخطيط النموذج نفسه، الذي يمكن العثور على كود "ثابت" مرة أخرى في قاموس "الرئيس" **Glob.html** عند بدء السكريبت. دراسة هذا "الرئيس": (انظر للصفحة 350) يخبرنا أنه سيتم هذه المرة استدعاء الأسلوب **reservations()** عندما يقوم المستخدم بالضغط على زر <*Enregistrer*> (حفظ). هذا الأسلوب يتلقى برامترات **spect** و **places** والقيم المدخلة من قبل الزائر في النموذج، ثم يقوم بتجميعها في مصفوفة مغلقة، ويضيفها إلى القائمة الموجودة في متغير الجلسة **"caddy"**. ثم يقوم بإرجاع صفحة صغيرة للمستخدم لتحديث طلباته.

السطر 148 : جميع البرامترات التي تم تمريرها عن طريق نموذج HTML هي سلاسل نصية. إذا كانت البرامترات رقمية، لابد من تحويلها إلى نوع مناسب قبل استخدامها على هذا النحو.

الأساليب التالية تسمح للمستخدم "عميل" بإغلاق زيارته للموقع وتسجيل الحجوزات، أو مراجعة الحجوزات التي قدمت سابقا. الأساليب للدالات المحجوزة لل "مديرين" تأتي الآن. جميع هذه الأساليب مبنية على نفس المبدأ ولا تتطلب التعليقات.



ينبغي أن تكون إستعلامات SQL الواردة في الأسطر التالية تفسيرية. سوف تتعرف على نوعين من الصلات.

> وصف الفصل لهذه الأسطر سيكون خارج نطاق الكتاب. فإذا كانت تبدو معقدة قليلا. لا تشعر بالإحباط : تعلم هذه اللغة يمكن أن يكون تدريجيا. اعلم أن هذا الأمر لا بد منه إذا كنت ترغب بأن تصبح مطورا ماهرا حقيقيا .

```
    def finaliser(self):
        # حفظ "caddy" للعميل في قاعدة البيانات.
        nom =cherrypy.session["nom"]
        mail =cherrypy.session["mail"]
        tel =cherrypy.session["tel"]
        caddy =cherrypy.session["caddy"]
        # حفظ المعلومات الخاصة بالعميل في جدول مخصص :
        req ="INSERT INTO clients(nom, e_mail, tel) VALUES(?,?,?)"
        res =BD.executerReq(req, (nom, mail, tel))
        # استرجاع مرجع تم تعيينه تلقائيا :
        req ="SELECT ref_cli FROM clients WHERE nom=?"
        res =BD.executerReq(req, (nom,))
        client =res[0][0]          # استخراج العنصر الأول من المصفوفة المغلقة الأول
        # تدوير caddy - تسجيل أماكن لكل مشاهد :
        for (spect, places) in caddy:
            # البحث عن آخر رقم مكان تم حجزه لهذا المشاهد. :
            req ="SELECT MAX(place) FROM reservations WHERE ref_spt =?"
            res =BD.executerReq(req, (int(spect),))
            numP =res[0][0]
            if numP is None:
                numP =0
            # توليد أرقام الأماكن التالية, حفظ :
            req ="INSERT INTO reservations(ref_spt,ref_cli,place) VALUES(?,?,?)"
            for i in range(places):
                numP +=1
                res =BD.executerReq(req, (spect, client, numP))
            # حفظ عدد الأماكن المباعة لهذا المشاهد :
            req ="UPDATE spectacles SET vendues=? WHERE ref_spt=?"
            res =BD.executerReq(req, (numP, spect))
        cherrypy.session["caddy"] =[]       # تفريغ لب caddy
        cherrypy.session["nom"] =""         # نسيان "الزائر"
        return mep("<h3>Session terminée. Bye !</h3>")
    finaliser.exposed =True

    def revoir(self):
        # العثور على الحجوزات التي تم تنفيذها من قبل عميل معين.
        # (نجد إشارته باستخدام عنوان بريد الإلكتروني) :
        mail =cherrypy.session["mail"]
        req ="SELECT ref_cli, nom, tel FROM clients WHERE e_mail =?"
        res =BD.executerReq(req, (mail,))
        client, nom, tel =res[0]
        # Spectacles pour lesquels il a acheté des places :
        req ="SELECT titre, date, place, prix_pl "\
             "FROM reservations JOIN spectacles USING (ref_spt) "\
             "WHERE ref_cli =? ORDER BY titre, place"
        res =BD.executerReq(req, (client,))
        # إنشاء جدول لقائمة المعلومات الموجودة html :
        tabl ='<table border="1" cellpadding="5">\n'
        tabs =""
        for n in range(4):
            tabs +="<td>{{{0}}}</td>".format(n)
        ligneTableau ="<tr>" +tabs +"</tr>\n"
        # أول صف من الجدول يحتوي على رؤوس الأعمدة :
        tabl += ligneTableau.format("Titre", "Date", "N° place", "Prix")
        # الصفوف التالية :
        tot =0                              # حساب السعر الأجمالي
        for titre, date, place, prix in res:
```

```
            tabl += ligneTableau.format(titre, date, place, prix)
            tot += prix
        # إضافة سطر في أسفل الجدول مع المجموع بشكل بارز :
        tabl += ligneTableau.format("", "", "Total", str(tot))
        tabl += "</table>"
        return mep(Glob.html["revoir"].format(nom, mail, tel, tabl))
    revoir.exposed =True

    def entrerSpectacles(self):
        # إيجاد قائمة المشاهدين الحالية :
        tabl =listeSpectacles()
        # إرجاع نموذج لإضافة مشاهد جديد :
        return mep(Glob.html["entrerSpectacles"].format(tabl))
    entrerSpectacles.exposed =True

    def memoSpectacles(self, titre ="", date ="", prixPl =""):
        # تخزين مشاهد جديد
        if not titre or not date or not prixPl:
            return '<h4>Complétez les champs ! [<a href="/">Retour</a>]</h4>'
        req ="INSERT INTO spectacles (titre, date, prix_pl, vendues) "\
             "VALUES (?, ?, ?, ?)"
        msg =BD.executerReq(req, (titre, date, float(prixPl), 0))
        if msg: return msg          # رسالة خطأ
        return self.index()         # الرجوع إلى الصفحة الرئيسية
    memoSpectacles.exposed =True

    def toutesReservations(self):
        # عرض الحجوزات التي أدلى بها كل عميل
        req ="SELECT titre, nom, e_mail, COUNT(place) FROM spectacles "\
             "LEFT JOIN reservations USING(ref_spt) "\
             "LEFT JOIN clients USING (ref_cli) "\
             "GROUP BY nom, titre "\
             "ORDER BY titre, nom"
        res =BD.executerReq(req)
        # إنشاء جدول لعرض المعلومات الموجودة html :
        tabl ='<table border="1" cellpadding="5">\n'
        tabs =""
        for n in range(4):
            tabs +="<td>{{{0}}}</td>".format(n)
        ligneTableau ="<tr>" +tabs +"</tr>\n"
        # الصف الأول من الجدول يحتوي على رؤوس الأعمدة :
        tabl += ligneTableau.\
            format("Titre", "Nom du client", "Courriel", "Places réservées")
        # الصفوف التالية :
        for tit, nom, mail, pla in res:
            tabl += ligneTableau.format(tit, nom, mail, pla)
        tabl +="</table>"
        return mep(Glob.html["toutesReservations"].format(tabl))
    toutesReservations.exposed =True

# === البرنامج الرئيسي ===
# فتح قاعدة البيانات - يتم إنشاؤها إذا كانت غير موجودة :
BD =GestionBD(Glob.dbName)
BD.creaTables(Glob.tables)
# تحميل "علامات" صفحات في قاموس عام :
chargerPatronsHTML()
# إعادة تكوين وبدء خادم الويب :
cherrypy.config.update({"tools.staticdir.root":os.getcwd()})
cherrypy.quickstart(WebSpectacles(), config ="tutoriel.conf")
```

في نهاية السكريبت، نجد كالعادة أسطر قليلة من البرنامج الرئيسي، والتي ستكون مسؤولة عن تمثيل الكائن **BD** للتواصل مع قاعدة البيانات، وتحميل "رؤساء" HTML في قاموس **Glob.html**، وبدء عمل خادم الويب Cherrypy لتضمين مرجع الصنف الرئيسي لمعالج الاستعلامات.

السطر 272 يضمن الدليل الجذر للموقع والذي هو الدليل الحالي .

"رؤساء" الـ HTML

"رؤساء" الـ HTML تستخدم من قبل السكريبت (كسلاسل تنسيق) في ملف نصي واحد (spectacles.htm)، ونحن سوف نعيد إنتاجه بالكامل أدناه :

```
[*miseEnPage*]
<html>
<head>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
<link rel=stylesheet type=text/css media=screen href="/annexes/spectacles.css">
</head>
<body>
<h1><img src="/annexes/python.gif">Grand Théâtre de Python City</h1>
{0}
<h3><a href="/">Retour à la page d'accueil</a></h3>
</body>
</html>
##########
[*pageAccueil*]
<form action="/identification" method=GET>
<h4>Veuillez SVP entrer vos coordonnées dans les champs ci-après :</h4>
<table>
<tr><td>Votre nom :</td><td><input name="nom"></td></tr>
<tr><td>Votre adresse courriel :</td><td><input name="mail"></td></tr>
<tr><td>Votre numéro de téléphone :</td><td><input name="tel"></td></tr>
</table>
<input type=submit class="button" name="acces" value="Accès client">
<input type=submit class="button" name="acces" value="Accès administrateur">
</form>
##########
[*accesAdmin*]
<h3><ul>
<li><a href="/entrerSpectacles">Ajouter de nouveaux spectacles</a></li>
<li><a href="/toutesReservations">Lister les réservations</a></li>
</ul></h3>
##########
[*accesClients*]
<h3>Bonjour, {0}.</h3>
<h4>Veuillez choisir l'action souhaitée :<ul>
<li><a href="/reserver">Réserver des places pour un spectacle</a></li>
<li><a href="/finaliser">Finaliser l'enregistrement des réservations</a></li>
<li><a href="/revoir">Revoir toutes les réservations effectuées</a></li>
</ul></h4>
##########
[*reserver*]
<h3>Les spectacles actuellement programmés sont les suivants : </h3>
<p>{0}</p>
<p>Les réservations seront faites au nom de : <b>{1}</b>.</p>
<form action="/reservations" method=GET>
<table>
<tr><td>La réf. du spectacle choisi :</td><td><input name="spect"></td></tr>
```

```
<tr><td>Le nombre de places souhaitées :</td><td><input name="places"></td></tr>
</table>
<input type=submit class="button" value="Enregistrer">
</form>
Remarque : les réservations ne deviendront effectives que lorsque vous
aurez finalisé votre "panier".
##########
[*reservations*]
<h3>Réservations mémorisées.</h3>
<h4>Vous avez déjà réservé {0} place(s) pour {1} spectacle(s).</h4>
<h3><a href="/reserver">Réserver encore d'autres places</a></h3>
N'oubliez pas de finaliser l'ensemble de vos réservations.
##########
[*entrerSpectacles*]
<h3>Les spectacles actuellement programmés sont les suivants :
{0}
Spectacle à ajouter :
<form action="/memoSpectacles">
<table>
<tr><td>Titre du spectacle :</td><td><input name="titre"></td></tr>
<tr><td>Date :</td><td><input name="date"></td></tr>
<tr><td>Prix des places :</td><td><input name="prixPl"></td></tr>
</table>
<input type=submit class="button" value="Enregistrer">
</form>
</h3>
##########
[*toutesReservations*]
<h4>Les réservations ci-après ont déjà été effectuées :</h4>
<p>{0}</p>
##########
[*revoir*]
<h4>Réservations effectuées par :</h4>
<h3>{0}</h3><h4>Adresse courriel : {1} - Tél : {2}</h4>
<p>{3}</p>
##########
```

مع هذا المثال المطور قليلا، نأمل أنك قد فهمت الفائدة من التعليمات البرمجية المنفصلة لبيثون وكودات HTML في ملفات منفصلة، كما فعلنا، حتى يحتفظ برنامجك بالحد الأقصى من قابلية القراءة. تطبيق الويب هو في الواقع مقصود به في كثير من الأحيان أن ينمو ويصبح أكثر تعقيدا مع مرور الوقت. لذا يجب عليك أن تضع كل الفرص بجانبك لتظل دائما منظما وسهل الفهم. باستخدام تقنيات حديثة مثل البرمجة الشيئية، وأنت ستكون بالتأكيد على الطريق الصحيح لتنمو بسرعة وتصبح مثمرًا للغاية .



تمارين

السكريبت السابق يمكن استخدامه لتنفيذ اختبارات المهارة الخاصة بك في العديد من المجالات .

1.17 كما هو موضح أعلاه، يمكننا تنظيم موقع ويب على شبكة الإنترنت عن طريق تقسيم إلى عدة أصناف. قد يكون من الحكمة الفصل بين أساليب "العملاء" و"الإدراة" للموقع في أصناف مختلفة .

2.17 كما هو، السكريبت لا يعمل بالشكل الصحيح إلا إذا كان المستخدم قد قام بتعبئة جميع الحقول المتوفرة. ولذلك سيكون من المفيد إضافة سلسلة من التعليمات للسيطرة على القيم التي تم ترميزها، مع رسائل خطأ للمستخدم عند الضرورة .

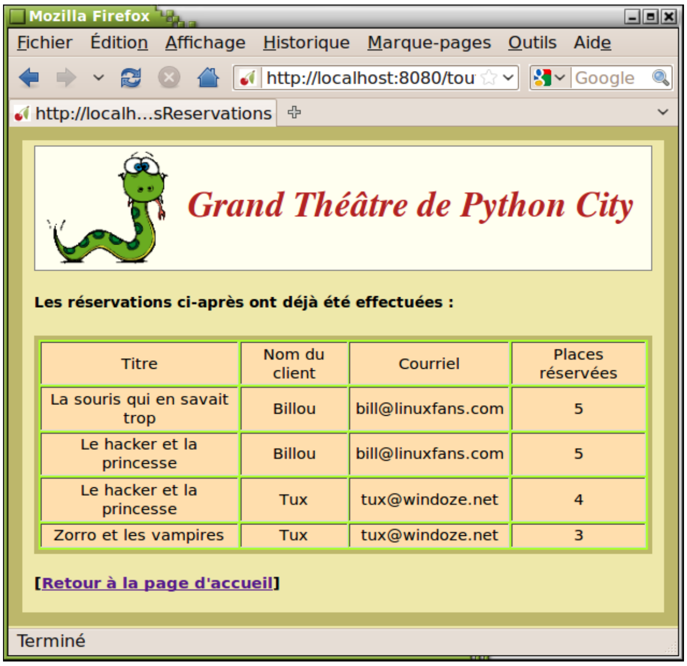
3.17 وصول المدراء يسمح فقط بإضافة عدد من عروض المسرحية للمسارح،لكن لا يمكن تعديل أو حذف تلك المشفرة بالفعل. أضف إذا أساليب لتنفيذ هذه المهام .

4.17 وصول المدير حر. قد يكون من الحكمة إضافة سكريبت آلي للتوثيق بكلمة المرور حتى يقتصر الوصول فقط للذين يعرفون السمسم (افتح يا سمسم) .

5.17 المستخدم "عميل" الذي يرتبط عدة مرات في كل مرة يتم تخزين فيها كعميل جديد، ومن ثم ينبغي أن يكون قادرًا على إضافة المزيد من الحجوزات إلى حسابه، وربما تعديل بياناته الشخصية، وما إلى ذلك، يجب أن يتم إضافة كل هذه المميزات .

6.17 ربما قد لاحظت أن جداول HTML يتم إنشاؤها من خلال سكريبت في دالة **listeSpectacles()**، من جهة، وفي أسلوب `revoir()` و **toutesReservations()** من جهة أخرى يتم إنشاء خوارزمية مشابهة جدا. سيكون من المثير للاهتمام أن نكتب دالة عامة قادرة على إنتاج مثل هذا الجدول، التي يصلنا وصفها في قاموس أو في قائمة. استلهم من دالة **listeSpectacles()** كقاعدة للبداية .

7.17 تزيين صفحات ويب تم إنشاؤها عن طريق سكريبت تم تعريفه في ملف CSS مرفق (spectacles.css). لا تتردد في تجربة ما يحدث إذا قمت بإزالة ورقة نمط الارتباط (السطر الخامس من ملف spectacles.htm) أو إذا قمت بتغيير محتوياته، والتي تصف أسلوب لتطبيقه على كل علامة .



تطويرات أخرى

إذا أردت جعل موقعك أكثر طموحا، يجب أن تأخذ عناء دراسة حزم أخرى مثل Pylons، Django، Karrigell، Plone، Zope، Twisted، TurboGears ... المرتبطة بنظم خادم أباتشي لنظام الخادم، و ySQL أو PostgreSQL لمعالجة قواعد البيانات. عليك أن تعرف أن هذا المجال واسع جدا، حيث يمكنك ممارسة إبداعاتك لفترة طويلة ...

18

الطباعة مع بيثون

عملنا جاهدين حتى الآن لتعلم كيفية تطوير تطبيقات يمكن استخدامها حقا. لمزيد من التقدم في هذا الاتجاه، سوف نكتشف الآن كيف يولد، مع هذه التطبيقات، مستندات قابلة للطباعة بجودة عالية. هذا يعني أننا سوف نكون قادرين على برمجة صفحات مطبوعة تحتوي على نصوص وصور ورسومات ...

> القليل من التاريخ
>
> عندما ظهرت أول أجهزة الحاسوب في منتصف القرن الماضي، فإنهم كانوا يفكرون في تقنيات الطباعة على ورقة النتائج التي تنتجها البرامج الحاسوبية. وفي وقت هذه الآلات، لم تكن الشاشات موجودة بعد، وكانت أساليب تخزين المعلومات بدائية جدا ومكلفة للغاية. لذلك تم اختراع الطابعات الأولى (بتكييف ال "*TTY*" المعاصرة) ليس فقط للتشاور، ولكن لتخزين البيانات المنتجة أيضا.
>
> هذه الآلات الأولى لم تكن تطبع في البداية سوى أرقام وأحرف كبيرة غير معلمة، والتي كان كافية في الوقت الذي كان يستخدم فيه أجهزة الحاسوب في حل المشاكل العلمية والحسابات للشركات الكبيرة.
>
> هذا الوضع لم يتغير كثيرا إلا أن ظهرت أجهزة الحاسوب الشخصية الأولى : في بداية الثمانينات[1]، في الواقع، الطابعات الأولى (عامة جدا) لا تستطيع طباعة سوى نص. ولغات البرمجة في ذلك الوقت كانت قادرة على إدارة الطباعة على الورق باستخدام بعض التعليمات البسيطة : في الواقع كانت تكفي أن تكون قادرة على شحن هذه الطابعات لطباعة سلاسل الأحرف، تتخللها بعض الرموز "غير المطبعية" المخصصة لأوامر خاصة : علامة تبويت، نهاية أسطر، نهاية صفحات ... إلخ
>
> هذه التعليمات بسيطة، على سبيل المثال تعليمة **`LPRINT`** للغة *BASIC*. تستخدم نفس الطريقة لعرض الحروف على الشاشة، أو حفظ أسطر النصوص في ملف (مثل الدالة **`print()`** أو أسلوب **`write()`** لبيثون).

الطابعات الحديثة متنوعة جدا وفعالة : معظم الطابعات الرسومية، التي تطبع أكثر مجموعات المحارف معرفة مسبقا في خط واحد، لكن مجموعات من النقاط الصغيرة من ألوان مختلفة. حتى لا يكون هنالك المزيد من التقييد يمكن للطابعة أن تكون :

محاذات الأحرف المطبعية المختلفة من أي أبجدية في مجموعة متنوعة بلا حدود للخطوط والأحجام، لكن أيضا للصور والرسومات من أي نوع، أو صور فتوغرافية.

من الواضح أن هذا ذا فائدة للمستخدم النهائي، لأنه يمكن الحصول على مجموعة من الإمكانيات الإبداعية الهائلة، ، ومع ذلك لم يجرؤ مبرمج على حلم الطباعة مرة واحدة ، هذا يعني أولا تعقيدا أكبر بكثير.

للتحكم في طباعة نص إلى طابعة حديثة، فإنه لا يكفي إرسال سلسلة من الأحرف من النص، كما كنا نحفظ النص نفسه. يجب علينا ان ننظر أن كل حرف هو رسم صغير تقوم الطابعة بإعادة رسمه نقطة نقطة في منطقة معينة من الصفحة. يجب أن يكون برنامج الحاسوب يهدف على إعداد تقارير مطبوعة من بين الأمور الأخر لمكتبة رموز[98] أو أكثر مقابلة للخطوط والأنماط التي نريد استخدامها، والخوارزميات الفعالة لترجمة هذه الرموز بمصفوفات من النقاط، مع دقة محددة وفي حجم معين.

لنقل كل هذه المعلومات، لابد من لغة حقيقية محددة. كما كنت تتوقع، يوجد العديد من هذه اللغات، مع وجود اختلاف في تحكم مختلف النماذج والعلامات التجارية من الطابعات في السوق .

يمكن للواجهة الرسومية المساعدة

في بداية هذا الكتاب، شرحنا باختصار أن هنالك تسلسل هرمي للغات الحاسوب، سواء لبرمجة جسم من التطبيق، ليقوم بإعادة إرسال استعلام لخادم قواعد البيانات، لوصف صفحة ويب، ... إلخ. أنت تعلم جيدا أن هنالك لغات ذات مستوى منخفض، يكون فيها تكوين الجمل غامضا في كثير من الأحيان، ومرهقة لأنها تتطلب استخدام تعليمات عديدة لأي أمر، لكن لحسن الحظ يوجد أيضا لغات ذات مستوى عال (مثل بيثون)، إن استخدام تكوين جملتها ممتع للغاية لأن يشبه (نوعا ما) اللغة البشرية (الإنكليزية في أغلب الأحيان)، وهي أكثر كفاءة بشكل عام بسبب تنفيذها السريع.

الذي يتحاور مع الطابعات الحديثة، هو نظام تشغيل الحاسوب. الذي يعالج باللغات منخفضة المستوى. فهي لا تستخدم سوى لكتابة برامج تسمى المترجمات الخاصة بالتحكم بالطابعات وهذه يتم تقديمها عادة من قبل الشركات المصنعة لهذه الطابعات، ومكتبات البرامج التي توفر واجهة مع لغات برمجة عامة أكثر.

لأسباب واضحة لتنسيق، مكتبات البرامج تسمح بصنع هذه التطبيقات مع واجهة رسومية تقوم بإظهار صور بجميع أنواعها في نوافذ الشاشة (نص منسق، صور نقطية، إلخ) هي غالبا ما تكون قادرة على إدارة تنفيذ صفحة الطباعة من نفس الصور. يتم دمج بعض من هذه المكتبات في نظام التشغيل نفسه (في حالة API لنظام ويندوز)، لكن هنالك اعتماديات أقل، والتي يجب عليك أن تعطي ما تفضله. أليس من المؤسف، بل تحد من قابلية برامج سكريبتات بيثون ؟

98يسمى رسم لمحرف بحرف رسومي (انظر أيضا إلى صفحة Error: Reference source not found).

مكتبات الواجهة الرسومية تتوفر للغة برمجتك. فهي توفر أصناف كائنات تسمح ببناء صفحات طباعة بمساعدة الويدجات، تشبه قليلا عندما تصنع نوافذ على الشاشة. مع Tkinter، على سبيل المثال، يمكنك إعداد رسم صفحة للطباعة داخل اللوحة، ثم إرسال تمثيلها إلى الطابعة، شرط أن هذه الطابعة "تفهم" لغة الطباعة PostScript (للأسف غير منتشرة بكثرة). مكتبات واجهة أخرى مثل WxPython أو PyQt تقدم المزيد من الإمكنيات.

و الميزة الرئيسية لتقنيات الطباعة أنها تستغل مكتبات واجهة المستخدم الرسومية، في الواقع أن في سكريبت بيثون تكتب أساس التحكم كله : لذلك يمكنك التأكد من أن صفحة الطباعة صنعت قطعة قطعة في الشاشة من قبل المستخدم، وتظهر كما ستظهر في الطباعة، وتبدأ في طباعة نفسها بالطريقة التي تناسبك.

هذه التقنية لديها بعض العيوب : قابلية نقلها لا تعمل في جميع أنظمة التشغيل، وتنفيذها يتطلب تعلم مفاهيم معقدة إلى حد ما (سياق الجهاز، إلخ). بالإضافة إلى ذلك، فإن هذا منهج يبين لك حدود عند النظر لإنتاج مستندات مطبوعة بحجم معين، مع نصوص تحتوي على فقرات متعددة، تتخللها أرقام، وتتحكم في اي واحدة منها بدقة في التخطيط، مع وجود اختلاف في النمط، والمسافات، وعلامات التبويب، وإلخ ...في بداية هذا الكتاب، شرحنا باختصار أن هنالك تسلسل هرمي للغات الحاسوب، سواء لبرمجة جسم من تطبيق، ليقوم بإعادة إرسال استعلام لخادم قواعد البيانات، لوصف صفحة ويب، ... إلخ. أنت تعلم جيدا أن هنالك لغات ذات مستوى منخفض، يكون فيها تكوين الجمل غامضا في كثير من الأحيان، ومرهقة لأنها تتطلب استخدام تعليمات عديدة لأي أمر، لكن لحسن الحظ يوجد أيضا لغات ذات مستوى عال (مثل بيثون)، إن استخدام تكوين جملتها ممتع للغاية لأن يشبه (نوعا ما) اللغة البشرية (الإنكليزية في أغلب الأحيان)، وهي أكثر كفاءة بشكل عام بسبب تنفيذها السريع.

الذي يتحاور مع الطابعات الحديثة، هو نظام تشغيل الحاسوب. الذي يعالج باللغات منخفضة المستوى. فهي لا تستخدم سوى لكتابة برامج تسمى المترجمات الخاصة بالتحكم بالطابعات وهذه يتم تقديمها عادة من قبل شركات المصنع لهذه الطابعات، ومكتبات البرامج التي توفر واجهة مع لغات برجمة عامة الأكثر.

لأسباب واضح لتنسيق، مكتبات البرامج تسمح بصنع هذه التطبيقات مع واجهة رسومية التي تقوم بإظهار صور بجميع أنواعها في نوافذ الشاشة (نص منسق، صور نقطية، إلخ) هي فالبا ما تكون قادرة على إدارة تنفيذ صفحة الطباعة من نفس الصور. يتم دمج بعض من هذه المكتبات في نظام التشغيل نفسه (في حالة API لنظام ويندوز)، لكن هنالك إعتماديات أقل، والتي يجب عليك أن تعطي ما تفضله. أليس من المؤسف، بل تحد من قابلية برامج سكريبتات بيثون ؟

مكتبات الواجهة الرسومية تتوفر للغة برمجتك. فهي توفر أصناف كائنات تسمح ببناء صفحات طباعة بمساعدة الويدجات، تشبه قليلا عندما تصنع نوافذ على الشاشة. مع Tkinter، على سبيل المثال، يمكنك إعداد رسم صفحة للطباعة داخل اللوحة،

ثم إرسال تمثيلها إلى الطابعة، شرط أن هذه الطابعة "تفهم" لغة الطباعة PostScript (للأسف غير منشرة بكثرة). مكتبات واجهة أخرى مثل WxPython أو PyQt تقدم المزيد من الإمكنيات.

و الميزة الرئيسية لتقنيات الطباع أنها تستغل مكتبات واجهة المستخدم الرسومية، هو في الواقع أن في سكريبت بيثون تكتب أساس التحكم كله : لذلك يمكنك التأكد من أنه صفحة الطباعة صنعت قطعة قطعة في الشاشة من قبل المستخدم، وتظهر كما ستظهر في الطباعة، وتبدأ في طباعة نفسها بالطريقة التي تناسبك.

هذه التقنية لديها بعض العيوب : قابلية نقلها لا تعمل في جميع أنظمة التشغيل، وتنفيذها يتطلب تعلم مفاهيم عابس إلى حد ما (سياق الجهاز، إلخ). بالإضافة إلى ذلك، فإن هذا منهج يبين لك حدود عند النظر لإنتاج مستندات مطبوعة بحجم معين، مع نصوص التي تحتوي على فقرات متعددة، تتخللها أرقام، وتتحكم في اي واحدة منها بدقة في التخطيط، مع وجود إخلاف في النمط، والمسافات، وعلامات التبويت، وإلخ ...

لغة تصف صفحة للطباعة. PDF

في الواقع، ترتبط هذه القيود بهذه المشكلة التي تحدثنا عنها : فإذا استخدمنا هذه التقنيات، مستوى اللغة المنخفض جدا يستخدم في التواصل مع الطابعة يتطلب منا إعادة إختراع في سكريبتاتنا آليات معقدة، في حين أنها قد صنعت وأهتبرت وصقلت من قبل فريق من المطورين المهرة.

> عندما كنا في الفصل 16 قمنا بشرح إدارة قواعد البيانات. على سبيل المثال، رأينا أنه من الأفضل أن توكل هذه المهمة المعقدة إلى نظام برمجي متخصص (محرك قاعدة البيانات، *SGBDR*)، بدلا من محاولة اختراع كل شيء في سكريبتاتنا. ويمكن لهذه التعليمات ببساطة توليد تعليمات للغة عالية المستوى (*SQL*) مناسبة بشكل مثالي لوصف الاستعلامات الأكثر تعقيدا، والسماح لـ *SGBDR* بفكها وتشغيلها على نحو أفضل .

بطريقة مختلفة قليلا، سوف نعرض لك في الصفحات التالية كيف يمكنك بسهولة بناء عن طريق بيثون بتعليماتها عالية المستوى لتصف وبقدر كبير من التفاصيل أن الطابعة يجب أن تظهر على صفحة مطبوعة. هذه اللغة هي **PDF** (تنسيق المستندات المحمولة) .

> وضعت أصل من قبل *Adobe Systems* في عام 1993 لوصف الوثائق التي سيتم طباعتها بطريقة مستقلة تماما من أي نظام برنامجي أو نظام تشغيل. الـ ***PDF*** يتم تقديمه في الكثير من الأحيان كتنسيق ملف، لأن عادة ما يتم إنشاؤه من خلال مكتبات دالات أو من خلال برامج متخصصة كسكريبتات كاملة. وهذا يعني لغة وصف للصفحات متطورة جدا، التي إكتسبت صفة معايير العالمية .

> في البداية كان خاصا. وأصبح هذا المعيار مفتوحا في عام 2008. لا تقلق بشأن متانته أو حرية استخدامه مع تطبيقاتك .

مستند PDF هو سكريبت يمكنه وصف النصوص والخطوط والأنماط والكائنات الرسومية وتخطيط من مجموعة من الصفحات للطباعة، التي يجب أن تعاد بنفس الطريقة، بغض النظر عن التطبيق والمنهاج المستخدم منصة لقراءته.

من مميزات لغة PDF أنها عامة غير مفهومة بسهولة من قبل الطابعات العادية. لتفسيرها فمن الضروري استخدام برامج متخصصة مثل Evince، Sumatra reader، Foxit reader، Acrobat Reader...

لا ينبغي أن تنظر إليه على أنه عيب، لأن هذه البرامج لديها ميزة كبيرة بالسماح لمعاينة النص للطباعة. يمكن لمستند PDF إذا أرشفته وإلخ. دون الحاجة إلى طباعة واقعية. كل هذه البرامج مجانية، ويوجد ما لا يقل عن واحد منهم في التكوين القياسي من أي جهاز حاسوب حديث.

هذا الفصل لا يهدف إلى شرح كيفية التحكم مباشرة بآلة الطباعة من سكريبت بيثون. بدلا من ذلك، سوف يظهر لك كم هو من السهل صنع مستندات PDF ذات جودة عالية باستخدام تعليمات قوية ومقروءة. هذا المنهج يضمن قابلية البرامج الخاصة بك في حين يسمح لهم بقدرات الطباعة واسعة النطاق، والتي سوف تناسب بشكل جيد التطبيقات التي تقوم بتطويرها للويب.

معظم ما نحتاجه متاح في وحدة بيثون موزعة بترخيص حر : مكتبة الأصناف **ReportLab**.

> في الواقع إن *ReportLab* هو اسم شركة لندنية تطور مجموعة من أصناف بيثون لتوليد وثائق *PDF* ذات جودة عالية برمجيا. وتقوم الشركة بتوزيع جميع مكتباتها لقاعدة النظام برخصة حرة. مما يسمح لك باستخدامها مجانا. وهي تقوم ببيع البرامج التكميلية لمعالجة *PDF* تحت رخصة ملكية مختلفة. وتطوير التطبيقات المخصصة للشركات. ولكن المكتبات الأساسية سوف تكون أكثر من كافية لجعلك سعيدا .

في هذه المرحلة من التفسيرات، ينبغي لنا أن نكون بالفعل قادرين على أن نقدم مثال سكريبت صغير يظهر لك كيفية استدعاء مكتبة ReportLab وصنع ملفات PDF بدائية. للأسف هذا غير ممكن في الوقت الحالي، لأننا يجب أن نحل مشكلة أولا : وحدة ReportLab الوحيدة المتاحة في وقت كتابة هذه السطر (كانون الأول/ديسمبر 2011) للإصدار 2 لبيثون. لا يوجد حتى الآن وحدة ReportLab لبيثون 3 !

إذا، ماذا نفعل ؟

تثبيت بيثون 2.6 أو 2.7 لاستخدام وحدات بيثون 2

في نهاية سنة 2008، قرر فريق تطوير بيثون لأول مرة كسر توافق الإصدارات المتتالية للغة بيثون من 2.5 إلى 3.0، وأعربوا عن أملهم أن هذا الإصدار الجديد سوف يعتمد عليه بسرعة من قبل مطوري مكتبات الطرف الثالث مثل ReportLab. لكن لسوء

الحظ، لم يكن هذا الحال : العديد من الملحقات متاحة منذ فترة طويلة لبيثون 2 وما تزال غير متواجدة لبيثون 3 (ليس فقط ReportLab، ولكن حتى Django، py2exe، PIL (Python Imaging Library)... إلخ). أنت تعرف أن هذا الوضع محرج، ونلاحظ أيضا أن العديد من الشركات لا تزال تفضل استخدام بيثون 2 على الرغم من عيوبها القليلة، بدلا من اعتماد الإصدار 3 وهو ما يتطلب منهم تحويل العديد من السكريبتات الموجودة، ولقد أصدر مطوروا بيثون بسرعة نسخة من التحول للغة، النسخة 2.6، ثم الثانية (من المفترض أن تكون الأخيرة) الإصدار 2.7.

هذان الإصداران من بيثون متوافقان تماما مع كافة الإصدارات السابقة، ولكنها تقبل تكوين جمل أدخلت في بيثون 3 حيثما كان ذلك ممكنا، بالإضافة إلى العديد من الوظائف الجديدة.

تخضع لتغييرات طفيفة في الأسطر الأولى، يتم تشغيل سكريبتات بيثون 3 تماما مثل الملفات 2.6 أو 2.7، والذي يسمح لهم باستخدام جميع مكتبات الجهات الأخرى غير المتاحة بعد لبيثون 3 !

لبقية هذا الفصل، فإنه نفترض بالإضافة إلى بيثون 3، مثبتة أيضا بيثون 2.6 أو 2.7 على محطة العمل الخاصة بك. لا تقلق من تثبيت بيثون 2 وبيثون 3 على نفس الجهاز / هذه الإصدارات مستقلة عن اللغة لا تعوق بعضها البعض .

> ليعلم النظام مع أي إصدار بيثون لتشغيل سكريتاتك. حدد ببساطة في بداية الأمر :
> ***python nom_du_script*** ليعمل ببيثون 2
> ***python3 nom_du_script*** ليعمل ببيثون 3 .

العديد من سكريبتات الفصول السابقة من هذا الكتاب تعمل دون تغيير تقريبا لبيثون 2.6 أو 2.7. على سبيل المثال، تقريبا جميع سكريبتات فصول 8، 13، 14، 15 (برمجة واجهة المستخدم الرسومية) تعمل بدون مشاكل إذا استبدلت ببساطة "t" الصغيرة في التعليمة: `from tkinter import *` بـ "T" كبيرة: `from Tkinter import *`

كل واحد من إصدارات بيثون لديه في الواقع إصدار خاص لمكتبة Tkinter، الاختلاف بين الأسماء متعمد لتجنب استدعاء وحدة نمطية عن طريق الخطأ عند استخدام إصداري بيثون (مع مكتبات Tkinter لكل واحدة منهم) موجود على نفس الجهاز.

كما أنه من السهل جدا تغيير هذه السكريبتات لتعمل على أي إصدار من الإصدارين. يكفي ببساطة تغيير بعض التعليمات في بداية البرنامج النصي للكشف عن أي إصدار بيثون يستخدم، وبالتالي يتم تغيير اللازم .

للقيام بذلك، يمكنك على سبيل المثال استخدام وحدة **sys** للمكتبة القياسية، بسمة **version** التي تشير إلى إصدار مفسر بيثون المستخدم لتشغيل السكريبت (في شكل سلسلة تحتوي على عدد قليل من المعلومات[99]). مظاهر:

99سلسلة تشير إلى رقم الإصدار الكامل (2.7.1+ في المثال أعلاه)، و رقم و تاريخ تجميع للمترجم، و رقم النسخة المستخدمة (GCC 4.5.2 في مثالنا) .

```
>>> import sys
>>> sys.version
'2.7.1+ (r271:86832, Apr 11 2011, 18:05:24) \n[GCC 4.5.2]'
```

في بداية سكريبتاتك، يجب أن تشمل الأسطر التالية :

```
# تحديد نسخة بيثون المستخدمة (2.xx أو 3.xx) :
import sys
if sys.version[0] =='2':       # الحرف الأول من السلسلة تكفينا
    from Tkinter import *      # لبيثون 2 وحدة Tkinter
else:
    from tkinter import *      # بيثون 3 وحدة tkinter
```

يمكنك أيضا استخدام إحدى التقنيات الأخرى الأكثر "بدائية"، على سبيل المثال محاولة استيراد مكتبات وراء تعليمة **try**:، وترتد إليك إذا لم تعمل (انظر إلى معالجة الاستثناءات، صفحة 125):

```
try:
    from tkinter import *
except:
    from Tkinter import *
```

مع الكود المصدري للأمثلة في هذا الكتاب الذي هو تحت تصرفك على الويب (انظر للصفحة ط)، نحن نقدم لك في الدليل الفرعي **py3onpy2** إصدارات "مختلطة" من سكريبتات الفصول 8 و 13 و 14 و 15 : والتي يمكنك تشغيلها إما تحت بيثون 3 أو بيثون 2.

جميع السكريبتات تستخدم واجهة المستخدم الرسومية Tkinter لمدخلاتها/مخرجاتها القابلة بسهولة على التكيف مع أي نسخة من بيثون، حقيقة بسيطة وهي أن هنالك إصدارات من مكتبة هذه الواجهة، وأنت تستخدم واحدة أو أخرى حسب الحالة. ومع ذلك فإن الحالة تتعقد إلى حد ما للسكريبتات التي تدير بنفسها المدخلات/المخرجات، وهذا معناه السكريبتات التي تستخدم الدالات مثل **input**() أو **print**() (وحدة التحكم بطرفية، لوحة المفاتيح/الشاشة) أو **open**() (قراءة أو كتابة حروف في ملف) .

لقد رأينا في الفصل 10 أن واحدة من أهم التغييرات في الإصدار 3 من بيثون هو إعادة تعرف نوع string كما يجري الآن في سلسلة أحرف Unicode، بدلا من سلسلة من البيتات (و التي تتطابق مع نوع byte). تواصل الحاسوب مع ملحقاته لا يزال أداؤه بالبايتات. ولذلك يجب على جميع التعليمات التي تتحكم في إخراج الأحرف تشمل الآن آلية ترميز وجميعها تدير آلية لفك أحرف المدخلات، وهذا الذي لم يكن في إصدارات بيثون "القديمة" .

يبقى فقط تكيف بسيط لسكريبتات بيثون 3 باستخدام هذه التعليمات، بحيث يمكن تشغيلها تحت بيثون 2.6 و 2. في كلا الإصدارين، هو من الممكن فرض توافقية من العديد من دالات الباثون 3 مع تعليمة استدعاء خاصة، لتكوين الجملة : **from _**

import __future__ *****. مع هذه التعليمة، فإن بيثون 2 سيقوم باستبدال ***** بما يعادلها من بيثون 3 ببساطة باستدعاء وحدة الخاص **__future__**.

لنبدأ، سوف نعمل على معالجة السلاسل النصية الحرفية (و هذا يعني سلاسل نصية تم تعريفها في السكريبت نفسه) ولتعامل على أنها سلاسل Unicode، وذلك باستخدام تعليمة :

```
from __future__ import unicode_literals
```

و سوف نضمن بعد ذلك أن الدالة **print()** لبيثون 3 يمكن استخدامها في تعليمة **print** لبيثون 2 (التي لا تزال وظيفية)، وذلك بإضافة التعليمة :

```
from __future__ import print_function
```

يجب أن تكون الاستدعاءات أعلاه في بداية السكريبت. لا يمكن وضعها في كتلة التعليمات التي تتبع الكشف عن نسخة بيثون المستخدمة. وهذه ليس مشكلة، لأنه بمجرد أن يمكنك تجاهلها إذا شغلت السكريبت في بيثون 3.

الدالة **input()** لبيثون 2 تعمل بشكل مختلف عن بيثون 3. ومع ذلك يمكننا إعادة تعريف بسهولة السكريبت نفسه، كما في التعليمات البرمجية أدناه :

```
1# import sys
2# if sys.version[0] =="2":
3#     encodage =sys.stdout.encoding
4#     def input(txt =""):
5#         print(txt, end="")
6#         ch =raw_input()
7#         return ch.decode(encodage)
```

الوحدة **sys** يتم استدعاؤها في السطر 1 تسمح بتحديد إصدار بيثون في كل مرة والترميز في نص الطرفية المستخدمة. سمته **stdout.encoding** تحتوي في الواقع على: **"cp1252"**، **"cp850"**، **"cp437"** أو **"UTF-8"** بعد أن تقوم بتشغيل السكريبت في طرفية سابقة. نافذة MSDOS أو نافذة IDLE (واجهة المستخدم الرسومية لبيثون) في ويندوز XP، أو على نظام تشغيل حديث .

دالة **input()** لبيثون 3 تعمل مثل دالة **raw_input()** في بيثون 2، لكنها تستخدم سلاسل نصية Unicode، في حين أن Python 2 تستخدم سلاسل نصية بايتات. وهذا يفرض علينا استخدام الدالة **print()** لعرض النص الذي تم وضعه كبرامتر لاستدعاء الدالة (السطر 5)، ولفك ترميز الإخراج (السطر 7) سلسلة بايتات التي سوف تتلقاها من لوحة المفاتيح في ترميز واضح، والتي يمكن لحسن الحظ تحديدها بواسطة السطر 3.

فيما يتعلق بعمليات القراءة/الكتابة في ملفات، أخيرا، يجب استبدال دالة open() القياسية لبيثون 2, التي لا تؤدي إلى أي معالجة على سلاسل البايتات المتنقلة, وإحدى برامج ترميز الوحدة, و التي تعمل مثل بيثون 3[100]:

```
from codecs import open
```

هذا إلى حد كبير ... على الأقل فيما يتعلق بالمفاهيم التي تمت مناقشتها في هذا الكتاب للمبتدئين. كما وسبق ذكره أعلاه، سوف تجد في الدليل الفرعي **py3onp2** تعليمات برمجية مفتوحة للكتاب، موجودة على الويب، وبعض الأمثلة على هذه السكريبتات "المختلطة" في أي بيثون 2 أو 3 (انظر على سبيل المثال حل تمرين 10.45) .

> باختصار، يمكنك الاستمرار في تعلم البرمجة باستخدام بيثون 3، دون أي خوف من أن تجد نفسك في طريق مسدود. إذا وجدت في أي من مشاريعك دالات مكتبات ليست متاحة بعد لبيثون 3، يمكنك بسهولة جدا ضبط سكريبتاتك لتنفيذها مؤقتا تحت بيثون 2.7 (أو حتى 2.6) واستغلال جميع مكتباتها، بانتظار نسخة بيثون 3 .

تشغيل مكتبة ReportLab

يمكننا الآن أن نكتب أول سكريبتاتنا لتوليد مستندات PDF. هذه السكريبتات تم كتابتها بتكوين جملة بيثون 3، كما يتم تنفيذ كافة البرامج النصية في هذا الكتاب، لكن يمكن ذلك (موقتا) في بيثون 2.6 أو 2.7. عندما تكون مكتبة ReportLab متاحة لبيثون 3، الأسطر التكييف في بداية كل سكريبت يمكن حذفها. (لتثبيت ReportLab، انظر إلى صفحة 413).

أول مستند PDF بدائي

```
### مسودة وثيقة PDF مبسطة مصنوعة باستخدام ReportLab ###
# السكريبت مكتوب ببيثون 3، لكن يمكن تشغيله ببيثون 2.6 أو 2.7، بما أن المكتبة غير متوفرة .

from __future__ import unicode_literals        # بدون فائدة في بيثون 3

# أستدعاء بعض عناصر مكتبة ReportLab :
from reportlab.pdfgen.canvas import Canvas     # صنف كائنات "لوحة"
from reportlab.lib.units import cm             # قيمة 1 سم في نقطة 1 بيكا
from reportlab.lib.pagesizes import A4         # أبعاد A4

# 1) اختيار اسم الملف لوثيقة سيتم صنعها :
fichier ="document_1.pdf"
# 2) تمثيل "كائن لوحة": ReportLab مرتبط لهذا الملف :
can = Canvas("{0}".format(fichier), pagesize=A4)
# 3) تركيب عناصر مختلفة على اللوحة :
texte ="Mes œuvres complètes"               # سطر للطباعة
can.setFont("Times-Roman", 32)              # اختيار الخط
posX, posY = 2.5*cm, 18*cm                  # موقعه على الورقة
can.drawString(posX, posY, texte)           # رسم النص على اللوحة
# 4) حفظ النتيجة في ملف PDF :
can.save()
```

[100] حول هذا الموضوع راجع : "التحويلات التلقائية عند معالجة الملفات"، في صفحة 143.

بعد تشغيل السكريبت، سوف تجد أن وثيقتك PDF الأولى في الدليل الحال، باسم `document_1.pdf`. وستكون بشكل *DIN*[101] مع جملة صغيرة في الوسط "Mes œuvres complètes" تم صنها بخط Times-Roman بـ 24 نقطة .

تحليل السكريبت الصغير يظهر لك أن ***مع ReportLab لديك لغة عالية المستوى لوصف صفحات مطبوعة***. يمكنك أن تختصر الكود في 4 أسطر فقط، باستخدام التعليمات المركبة ![102]

تعليقات

- السطر 5 : التعليمة `from __future__ import unicode_literals` في بداية السكريبت تفرض مفسر بيثون 2 بتحويل السلاسل الحرفية في السكريبت إلى سلاسل Unicode (كما في بيثون 3). بدون هذه التعليمة، سيتم ترميز السلاسل البايتات وفقا لمعيار المستخدم لمحرر النص الخاص بك[103]. إذا كان المعيار ليس Utf-8 (على سبيل المثال،في حالة ويندوز XP)، هذه السلاسل لا يمكن أن تكون مقبولة من قبل مكتبة ReportLab. وهذا لا يقبل في الواقع إلا سلاسل Unicode أو سلاسل بايتات مشفرة بـ Utf-8 (إما).
- السطر 8 : مكتبة ReportLab كبيرة. فنحن لن نستدعي سوى العناصر الضرورية لهذا العمل. من أهم الأصناف هي صنف **Canvas()**، التي لديها قدرة هائلة من الأساليب للتخلص من أي شيء على صفحة مطبوعة : شظايا النص والفقرات والرسومات المتجهة، والصور النقطية ... إلخ.
- الأسطر 9 و 10 : الاستدعاءات هنا هي ببساطة قيم لتحسين إمكانية قراءة الكود. وحدة القياس في ReportLab هي النقطة المطبعية بيكا، الذي هي 1/72 بوصة، هو 0.353 مم. `A4` هي مصفوفة مغلقة بسيطة (595.28, 841.89) تعبر عن حجم DINA4 في هذه الوحدة، و `cm` لقيمة سنتيمتر (28.346)، التي من شأنها أن تستخدم على نطاق واسع في أمثلتنا، للتعبير عن أبعاد أكثر وضوحا ومواقع على الصفحة.
- السطر 15 : تمثيل كائن لوحة. يجب أن يوفر المنشئ اسم الملف الذي سيتلقى الوثيق، جنبا إلى جنب، ربما مع البرامترات الاختيارية. حجم الصفحة الافتراضية هي A4، لكن ليس سيئا بتحديد بشكل واضح .
- السطر 18 : الأسلوب **setFont()** يستخدم لتحديد خط الكتابة وحجمه. واحدة من نقاط القوة لـ PDF يكمن في حقيقة أن جميع البرامج لتفسرها (Acrobat Reader ...إلخ) يجب أن تكون مكتبة الرموز نفسها للخطوط التالية : Courier، Courier-Bold، Courier-BoldOblique، Courier-Oblique، Helvetica، Helvetica-Bold، Helvetica-BoldOblique، Helvetica-Oblique، Symbol، Times-Bold، Times-BoldItalic،

101 Le DIN (*Deutsches Institut für Normung*) **هو المعيار الوطني الألماني.**

102 **بالجمع بين الأسطر 11 و 13 من جهة و الأسطر 15 و 17 و 18 من جهة أخرى.**

103 **انظر إلى صفحة** Error: Reference source not found **: «مشاكل ممكنة مع الأحرف المعلمة».**

ZapfDingbats، Times-Roman، Times-Italic. هذه الخطوط تستخدم للعديد من الاستخدامات : هي في الواقع خط أحادي المسافة (Courier)، وخطين مناسبين بدون ترقيق (Helvetica، Times-Roman)، كل واحد منها لديه أربعة أنماط مختلفة (عادي، كبير، مائل، كبير ومائل) وأخيرا خطين من الرموز المختلفة (Symbol، ZapfDingbats). الحقيقة أن بالفعل "يعرف" برنامج المفسر يعفيك من وصفها في المستند نفسها : إذا أردت يمكنك الحصول على هذه الخطوط، ومستندات ال PDF والاحتفاظ بحجم صغير جدا .

> أداء تضمين المزيد من الخطوط المتجهة ؟
>
> من الممكن تماما استخدام خطوط أخرى متجهة (*TrueType* أو *Adobe Type 1*). لكن يجب بعد ذلك توفير وصف رقمي في الوثيقة نفسها. مما يزيد بشكل كبير من حجم وتعقيد بعض الأشياء. ونحن لن نشرح ذلك في هذا الكتاب .

- السطر 20 : الأسلوب **drawString(x, y, t)** يحدد موقع جزء النص **t** في اللوحة بمحاذات على يسار نقطة الإحداثيات **x, y**. المهم جدا أن نلاحظ هنا أنه ينبغي تحديد هذه الإحداثيات في الزاوية اليسرى السفلى للصفحة، كما هي العادة في الرياضيات، وعادة لا يتم الاعتماد على الزاوية أقصى اليسار لإحداثيات الشاشة. فإذا أردت كتابة العديد من أسطر النص المتتالي في الصفحة، يجب عليك التأكد من أن الإحداثيات العمودية تقلل من الأول إلى الأخير .
- السطر 22 : الأسلوب **save()** تكمل العمل وتغلق الملف. وسوف نرى لاحقا كيفية توليد مستندات متعددة الصفحات.

توليد مستند أكثر تفصيلا

السكريبت التالي يولد مستند غريب قليلا، لتسليط الضوء على بعض الاحتمالات الكثيرة التي أصبحت الآن متاحة لنا، بما في ذلك إعادة إنتاج صور نقطية على الصفحة. فإذا كنت ترغب في استخدام هذه الميزة، يجب أن تثبت مكتبة معالجة الصور **Python Imaging Library** (*PIL*)، وهي أداة ملحوظة يمكن أن تكون كائن كتاب وحدها. لاحظ أن المستندات المفصلة لـ ReportLab و PIL متاحة على الويب، على الأقل باللغة الإنكليزية. شرح تثبيت PIL موجود في الصفحة 413.




```
# === إنشاء وثيقة PDF مع انواع مختلفة من المسارات ===

# تكييف السكريبت لجعله يعمل مع بيثون 2.6 أو 2.7 :
# (لبيثون 3 Reportlab يمكنك حذف هذه الأسطر إذا تم توفير)
from __future__ import unicode_literals
from __future__ import division                       # "تقسيم" حقيقي
# ----------------------------------------------------------------------

# استدعاء بعض عناصر مكتبة ReportLab :
from reportlab.pdfgen.canvas import Canvas
from reportlab.lib.units import cm
from reportlab.lib.pagesizes import A4

fichier ="document_2.pdf"
can = Canvas("{0}".format(fichier), pagesize=A4)
# تركيب عناصر مختلفة على اللوحة :
largeurP, hauteurP = A4                               # عرض وارتفاع الصفحة
centreX, centreY = largeurP/2, hauteurP/2             # إحداثيات منتصف الورقة
```

```
19# can.setStrokeColor("red")                         # لون الأسطر
20# # تذكير : يتم حساب الموقع العمودي للصفحة من الأسفل.
21# can.line(1*cm, 1*cm, 1*cm, 28*cm)                  # خط عمودي على اليسار
22# can.line(1*cm, 1*cm, 20*cm, 1*cm)                  # خط أفقي في الأسفل
23# can.line(1*cm, 28*cm, 20*cm, 1*cm)                 # خط مائل (تنازليا)
24# can.setLineWidth(3)                                # سمك جديد للخطوط
25# can.setFillColorRGB(1, 1, .5)                      # لون التعبئة (RVB)
26# can.rect(2*cm, 2*cm, 18*cm, 20*cm, fill=1)         # مستطيل 18 x 20 سم
27#
28# # يقوم بإرجاع إحداثيات الرسم dramImage رسم نقطي (محاذات الزاوية اليسرى السفلي). لأسلوب
    النقطي (بالبيكسلات) في مصفوفة مغلقة :
29# dX, dY =can.drawImage("cocci3.gif", 1*cm, 23*cm, mask="auto")
30# ratio =dY/dX                                       # تقرير عرض\إرتفاع الصورة
31# can.drawImage("cocci3.gif", 1*cm, 14*cm,
32#               width=3*cm, height=3*cm*ratio, mask="auto")
33# can.drawImage("cocci3.gif", 1*cm, 7*cm, width=12*cm, height=5*cm, mask="auto")
34#
35# can.setFillColorCMYK(.7, 0, .5, 0)                 # لون التعبئة (CMJN)
36# can.ellipse(3*cm, 4*cm, 19*cm, 10*cm, fill=1)      # بيضاوي (! محاور = 16 x 6 سم)
37# can.setLineWidth(1)                                # سمك جديد للخطوط
38# can.ellipse(centreX -.5*cm, centreY -.5*cm,        # دائرة صغير تشير إلى
39#             centreX +.5*cm, centreY +.5*cm)        # موقع منتصف الصفحة
40#
41# # بعض النصوص ,مع خطوط وألوان متجهة ومتحاذية مختلفة :
42# can.setFillColor("navy")                           # لون النصوص
43# texteC ="Petite pluie abat grand vent."            # نص في الوسط
44# can.setFont("Times-Bold", 18)
45# can.drawCentredString(centreX, centreY, texteC)
46# texteG ="Qui ne risque rien, n'a rien."            # نص محاذاة على اليسار
47# can.setFont("Helvetica", 18)
48# can.drawString(centreX, centreY -1*cm, texteG)
49# texteD ="La nuit porte conseil."                   # نص محاذاة على اليمين
50# can.setFont("Courier", 18)
51# can.drawRightString(centreX, centreY -2*cm, texteD)
52# texteV ="L'espoir fait  vivre."                    # وضع النص عمودي
53# can.rotate(90)
54# can.setFont("Times-Italic", 18)
55# can.drawString(centreY +1*cm, -centreX, texteV) # ! قلب الإحداثيات !
56# texteE ="L'exception confirme la règle"            # نص لإظهاره باللون الأبيض
57# can.rotate(-90)                                    # العودة إلى الإتجاه الأفقي
58# can.setFont("Times-BoldItalic", 28)
59# can.setFillColor("white")                          # لون جديد للنصوص
60# can.drawCentredString(centreX, 7*cm, texteE)
61#
62# can.save()                                         # حفظ النتيجة
```

تعليقات

- السطر 6 : في بيثون 2، عامل قسمة هو / ينفذ قسمة عدد صحيح بشكل افتراضي (مقابلة لمعامل // في بيثون 3 - انظر للصفحة12). سيتم هنا فرض وضع القسمة الحقيقية .
- الأسطر من 17 إلى 40 : هذه الأسطر تظهر لك بعض الأساليب التي تسمح برسم رسوم على الصفحة باستخدام الأسطر والأشكال الأساسية. من الواضح، لا يمكننا تبسيطه أكثر هنا. فإذا أخذت عناء مراجعة المستندات المرجعية لـ

ReportLab (كتيبات PDF عديدة)، سوف تجد الكم الهائل من الأساليب الأخرى التي تسمح بتحقيق رسوم متجه لتحجيمها، والرسوم البيانية والجداول والرسوم بيانية الدائري ... إلخ .

• الأسطر من 19 إلى 23 تعرف مثلثا يلخص محيط أحمر. الأسطر من 24 إلى 26 ترسم محيطا مستطيلا أكثر سماكة وتملؤه بلون أصفر شاحب. لاحظ أن ترتيب التعليمات أمر بالغ الأهمية : الرسومات التي تتداخل على التوالي. لاحظ مرة أخرى أن جميع الإحداثيات العمودية تقوم بالصعود، بداية من أسفل الصفحة .

• الأسطر 19 و 25 و 36 و 43 و 60 : في ReportLab يمكن تحديد الألوان بثلاثة طرق مختلفة. وأبسط هذه الطرق هي استخدام أسماء الألوان باللغة الإنكليزية، لكن هذا لا يمكننا من اختيار الفروق الدقيقة الخاصة. يمكننا تحديد هذا عن طريق 3 مركبات تشير لألوان الضوء وهي الأحمر والأخضر والأزرق (باللغة الإنكليزية RGB) كما هو الحال لبكسلات الشاشة، أو أفضل من ذلك، لأنه صفحة مطبوعة، فهي تشير إلى 4 ألوان وهي سماوي وأرجواني وأصفر وأسود (باللغة الإنكليزية CMYK) وهي الأحبار المستخدمة لإنتاج صورة على ورق ، ويجب أن تكون قيم كل مكون بين 0 و 1.

• الأسطر من 28 إلى 34 : يستخدم الأسلوب **drawImage()** لإعادة إنتاج نفس صورة الخنفساء 3 مرات، في مواقع وأحجام مختلفة. هذا الإجراء ممكن بفضل دالات PIL. لاحظ أنه إذا استخدمت نفس الصورة عدة مرات في مستند، لن يتم تحميلها سوى مرة واحدة في PDF، عبر آلية تخزين مؤقت.

• السطر 30 : سوف نقوم بإعادة إنتاج الصورة النقطية أول مرة دون تغيير حجمها. سوف يعتبر ReportLab أنه في هذه الحالة كل بكسل من الصورة يساوي (1/72 بوصة). البرامتر الأول يمرر إلى الأسلوب **drawImage()** اسم الملف الذي يحتوي على الصورة، الصيغ المقبولة هي TIF، GIF ، JPG ، PNG و BMP. البرامترات الثانية والثالث هي إلزامية فهي تحدد إحداثيات الركن الأيسر السفلي على الصفحة. البرامتر الاختيار `mask="auto"` مطلوب إذا كنت تريد أن تجعل الصورة شفافة[104].

• الأسطر من 31 إلى 33 : ميزة مفيدة لأسلوب **drawImage()** فهي تقوم بإرجاع أبعاد الصورة النقطية بالبكسل، في نفق من الأعداد الصحيحة. يمكنك إذا استخدام هذه المعلومات في سكريبت خاص بك، وكما هو الحال هنا لتحديد طول وارتفاع الصورة، لإعادة رسمها بالإضافة إلى مستوى آخر، وذلك بفضل البرامترات الاختيارية **width** و **height**.

• الأسطر من 34 إلى 40 : نعيد رسم الصورة مرة أخرة، وهذه المرة مع أي أبعاد، ثم نغطيها جزئيا بقوس بيضوي. لاحظ الاختلاف بين الأسلوب **rect()** في السطر 26، الأسلوب **ellipse()** يحتاج إلى 4 برامترات والتي هي إحداثيات x و y للزوايا اليسرى السفلية واليمنى العلوية للمستطيل الذي سيتم وضع به القوس البيضوي، وللأسلوب **rect()**، هذه 4 برامترات هي أحداثيات x و y للزاوية اليسرى للمستطيل، ثم عرضه وارتفاعه. والأسطر من 38 إلى 40 ترسم دائرة

---

104 **يمكن لهذا "القناع" أن يحتوي على قيم أخرى، لكن من الواضح أن هذا لا يمكن وضعه في مقدمة عن Reportlab المحدودة جدا. نحن لا يمكننا أن نسمح بتفاصيل أكثر لكل خيار أو برامتر مقدم .**

صغيرة في وسط الصفحة، والتي ستكون بمثابة مؤشر لتحديد مواقع لفهم بعض الأسطر من النص التي ترسم بتعليماتنا الأخيرة .

- الأسطر من 42 إلى 61 : يرجى النظر إلى هذه الخطوط. سوف نظهر لكم كيف يمكنكم المحاذاة إلى اليسار، وإلى اليمين وفي وسط خط النص، باستخدام أساليب **drawRightString**()، **drawString()** و **drawCentredString**(). يمكنك بالطبع استخدام خطوط وألوان وأحجام مختلفة للأحرف، وحتى تدوير النص في أي زاوية !

مستندات متعدد الصفحات وإدارة الفقرات

Gestion des paragraphes avec ReportLab

La **programmation** est l'art d'apprendre à une machine comment accomplir de nouvelles tâches, qu'elle n'avait jamais été capable d'effectuer auparavant.

C'est par la programmation que vous pourrez acquérir le plus de contrôle, non seulement sur votre machine, mais aussi peut-être sur celles des autres par l'intermédiaire des réseaux. D'une certaine façon, cette activité peut donc être assimilée à une forme particulière de magie.

Elle donne effectivement à celui qui l'exerce un certain pouvoir, mystérieux pour le plus grand nombre, voire inquiétant quand on se rend compte qu'il peut être utilisé à des fins malhonnêtes.



Dans le monde de la programmation, on désigne par le terme ***hacker*** les programmeurs chevronnés qui ont perfectionné les systèmes d'exploitation de type Unix et mis au point les techniques de communication qui sont à la base du développement extraordinaire de l'Internet.

Ce sont eux également qui continuent inlassablement à produire et à améliorer les logiciels libres (*Open Source*).



Selon notre analogie, les hackers sont donc des maîtres-sorciers, qui pratiquent la magie blanche.

Mais il existe aussi un autre groupe de gens que les journalistes mal informés désignent erronément sous le nom de *hackers*, alors qu'ils devraient plutôt les appeler *crackers*.

Ces personnes se prétendent *hackers* parce qu'ils veulent faire croire qu'ils sont très compétents, alors qu'en général ils ne le sont guère.

Ils sont cependant très nuisibles, parce qu'ils utilisent leurs quelques connaissances pour rechercher les moindres failles des systèmes informatiques construits par d'autres, afin d'y effectuer toutes sortes d'opérations illicites : vol d'informations confidentielles, escroquerie, diffusion de spam, de virus, de propagande haineuse, de pornographie et de contrefaçons, destruction de sites web, etc.

Ces sorciers dépravés s'adonnent bien sûr à une forme grave de magie noire.



Mais il y en a une autre.

Les vrais *hackers* cherchent à promouvoir dans leur domaine une certaine éthique, basée principalement sur l'émulation et le partage des connaissances. La plupart d'entre eux sont des perfectionnistes, qui veillent non seulement à ce que leurs constructions logiques soient efficaces, mais aussi à ce qu'elles soient élégantes, avec une structure parfaitement lisible et documentée.

Vous découvrirez rapidement qu'il est aisé de produire à la va-vite des programmes qui fonctionnent, certes, mais qui sont obscurs et confus, indéchiffrables pour toute autre personne que leur auteur (et encore !).

Cette forme de programmation absconse et ingérable est souvent aussi qualifiée de « magie noire » par les *hackers*.

La démarche du programmeur

Comme le sorcier, le programmeur compétent semble doté d'un pouvoir étrange qui lui permet de transformer une machine en une autre, une machine à calculer en une machine à écrire ou à dessiner, par exemple, un peu à la manière d'un sorcier qui transformerait un prince charmant en grenouille, à l'aide de quelques incantations mystérieuses entrées au clavier.

Comme le sorcier, il est capable de guérir une application apparemment malade, ou de jeter des sorts à d'autres, via l'Internet.

Mais comment cela est-il possible ?

في هذا الفصل المقدم لوظائف ReportLab، يمكننا لمس موضوع واسع جدا. والذي قمنا بشرحه بإيجاز في الصفحات السابقة للطبقة الأدنى من هذه المكتبات، مستوى "اللوحة". فوق الطبقة الأولى، يوجد أربعة طبقات أخرى:

- العناصر "fluables"[105] : والتي هي أهم الفقرات (أجزاء من النص المنسق)، لكن يمكن أن تكون صور ومسافات وجداول ...إلخ، والتي تشترك في أنها يمكنك "إلقاء" بعضها واحد تلو الأخر في مناطق محددة مسبقا من المستند.
- الإطارات، التي تحدد المناطق المستطيلة على الصفحة، حيث يمكنك "تدفق" العناصر المختلفة fluables التي تم شرحها فوق، في تدفق مستمر.
- أنماط الصفحة، التي هي نماذج مختلفة من الصفحات مع رؤوس وتذييلات، وإطارات وترقيم.
- أنماط المستندات (أو النماذج)، التي تقدم مخططات مختلفة محددة مسبقا .

عند قراءة ما سبق، عليك أن تعرف أن تصرفك مع ReportLab هي أداة محترفة لمستوى عال، يقدم لك كل ما هو متوقع من نظام معالج نصوص حديث. بدلا من ذلك نحن نفتقر إلى الوصف كاملا، ولكن سنحاول شرح آليات تنفيذها على المستويات الرئيسية 2 و 3، و fluables والإطاران.

> يجب على تفسيراتنا أن تكون كافية لإعداد مستنداتك الأولى. ولكننا نشجع وبقوة أن تقرأ الوثائق واسعة النطاق على شبكة الإنترنت على الموقع الرسمي لـ *Reportlab*. من أجل تحقيق أقصى فائدة .

مثال عن سكريبت تخطيط ملف نصي

السكريبت التالي تحمل أسطر لملف نصي بسيط في قائمة. كل سطر يتم تحويله إلي كائن fluable من نوع "فقرة". يتم إنشاء fluable أخرى ويتم إدراجها بين الفقرات : تباعد بين كل واحدة منهم، ومن وقت لأخر تنسيق الصورة. يتم وضع كل هذه الكائنات في قائمة واحدة، والتي تستخدم لعد 1ك في تدفق مصادر الطاقة لملئ 5 إطارات (frames). يتم التعامل مع الفائض المتاح في نهاية مختلفة لتسليط الضوء على بعض الأساليب الأساسية جدا لكائنات-الفقرات .

```
# === توليد وثيقة PDF مع معالج fluables (الفقرات) ===

# تكييف السكريبت لجعله يعمل مع بيثون 2.6 أو 2.7 :
# (لبيثون 3 Reportlab يمكنك حذف هذه الأسطر إذا تم توفير)
from __future__ import unicode_literals
from __future__ import division                # "تقسيم" حقيقي
from codecs import open                        # ترميز ملفات النصية
# ------------------------------------------------------------------------

# استدعاء بعض عناصر مكتبة ReportLab :
from reportlab.pdfgen.canvas import Canvas
from reportlab.lib.units import cm
from reportlab.lib.pagesizes import A4
from reportlab.platypus import Paragraph, Frame, Spacer
from reportlab.platypus.flowables import Image as rlImage
from reportlab.lib.styles import getSampleStyleSheet
```

105 مصطلح Fluable : "الذي يتدفق" (من اللاتينية fluere) . صفة نادر ما تستخدم في اللغة الحالية، تم إستخدامها هنا كترجمة تعبير جديد تقريبي من الإنجليزية flowable من الفعل to flow التدفق) و الذي يعني هنا : "عناصر تدفق كيانات قابلة للطباعة" .

```

# إنشاء قائمة لسلاسل نصية لتحويلها + loin en paragraphes :
ofi =open("document.txt", "r", encoding="Utf8")
txtList =[]
while 1:
    ligne =ofi.readline()
    if not ligne:
        break
    txtList.append(ligne)
ofi.close()

# === صنع وثيقة PDF :
fichier ="document_3.pdf"
can = Canvas("{0}".format(fichier), pagesize=A4)
styles = getSampleStyleSheet()             # قاموس أنماط معرفة مسبقا
styleN =styles["Normal"]                   # كائن صنف ParagraphStyle()

# عناصر تسمى الفقرات والأرقام والمباعدات "fluables".
# إضافة هذه عناصر fluables في قائمة <story> ("تاريخ") :
n, f, story = 0, 0, []
for txt in txtList:
    story.append(Paragraph(txt, styleN))      # إضافة فقرة
    n +=1                                     # عد الفقرات المولدة
    story.append(Spacer(1, .2*cm))            # إضافة مساحة (تباعدة)(2مم)
    f +=2                                     # عد الـ fluables المولَّدة
    if n in (3,5,10,18,27,31):                # إضافة صورة نقطية
        story.append(rlImage("cocci3.gif", 3*cm, 3*cm, kind="proportional"))
        f +=1

# === إعداد الصفحة الأولي :
can.setFont("Times-Bold", 18)
can.drawString(5*cm, 28*cm, "Gestion des paragraphes avec ReportLab")
# وضع ثلاثة إطارات (2 "أعمدة" وواحد "أسفل الصفحة") :
cG =Frame(1*cm, 11*cm, 9*cm, 16*cm, showBoundary =1)
cD =Frame(11*cm, 11*cm, 9*cm, 16*cm, showBoundary =1)
cI =Frame(1*cm, 3*cm, 19*cm, 7*cm,  showBoundary =1)
# وضع عناصر الـ fluables في هذه الأطر الثلاثة :
cG.addFromList(story, can)                    # ملء الإطار على اليسار
cD.addFromList(story, can)                    # ملء الإطار على اليمين
cI.addFromList(story, can)                    # ملء الإطار السفلي

can.showPage()                                # الانتقال إلى الصفحة التالية

# === إعداد الصفحة الثانية :
cG =Frame(1*cm, 12*cm, 9*cm, 15*cm, showBoundary =1)       # إطارات
cD =Frame(11*cm, 12*cm, 9*cm, 15*cm, showBoundary =1)      # (= 2 عمودان)
cG.addFromList(story, can)                    # ملء الإطار على اليسار
cD.addFromList(story, can)                    # ملء الإطار على اليمين

# معالجة فردية لعناصر الـ fluables المتبقية :
xPos, yPos = 6*cm, 11.5*cm                    # موقع البدء
lDisp, hDisp = 14*cm, 14*cm                   # العرض والارتفاع المتاح
for flua in story:
    f += 1
    l, h =flua.wrap(lDisp, hDisp)             # العرض والارتفاع الفعلي
    if flua.identity()[1:10] =="Paragraph":
        can.drawString(2*cm, yPos-12, "Fluable n° {0}".format(f))
    flua.drawOn(can, xPos, yPos-h)            # تثبيت fluable
    yPos -=h                                  # موقع التالي

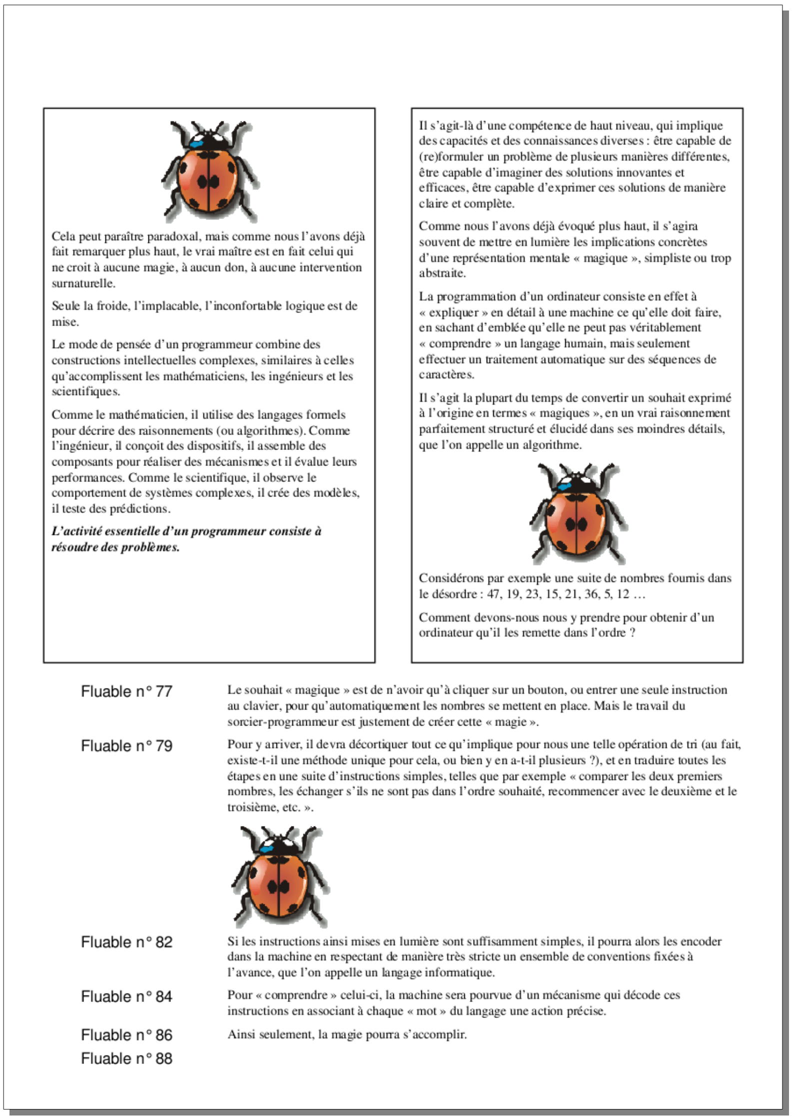
can.save()                                    # إنهاء الوثيقة
```

تعليقات

- السطر 14 : يوفر ReportLab صنف الإطارات (frames)، العديد من أصنف كيانات fluables يمكنهم "الإلقاء" في تدفق مستمر. نحن لا نستخدم هنا إلا fluables من نوع فقرة ومسافات وصورة. بالمناسبة استخدام as لإعادة تسمية الصنف **Image()** المستدعى : هذا العمل الوقائي مفيد لتفادي الخلط مع صنف **Image()** من مكتبة تصوير بيثون (PIL) التي يمكن أن تستخدم أيضا ربما في السكريبت أيضا.
- السطر 16 : الصنف **Paragraph()** يخصص فصلا كاملا. كائنات المثيل من هذا الصنف تسمح بتخطيط دقيق لكل جزء من أجزاء النص، ينسق وفقا لرغباتك في نمط معين. بعض الأنماط الأساسية متوفرة في وحدة `reportlab.lib.styles`. سمة `getSampleStyleSheet` يتم تنظيمها مثل القاموس، ولكن الشيء المهم هو أنه يمكنك بسهولة تعديل أي منها لتناسب احتياجاتك عن طريق تعديل خصائصه المختلفة الافتراضية (الخط واللون، ثم السحب أو/و المسافات، تبويبات، إضافة تعداد بوصي أو رقمي، المحاذات، إلخ). ويقترح مثال على تعديل نمط الفقرة في وقت لاحق، في التمرينات في نهاية هذا الفصل.
- الأسطر من 18 إلى 26 : مصدر فقراتنا هو ملف نص بسيط، نحن سنقوم باستخراج سلاسل نصية بالطريقة العادية (انظر صفحة 143). يرجى ملاحظة أن هذا النص يحتوي على عدد من علامات التنسيق على الإنترنت من نوع XML، مثل على سبيل المثال <u> و</u> لجعل جزء من النص تحته سطر، أو <i> و</i> لجعله مائلا، إلخ). نحن لا يمكننا أن نقدم هنا قائمة جميع العلامات. لاحظ أنه لا يمكنك استخدام رموز محجوزة مثل > و& في النص الذي سيتحول إلى فقرة ReportLab .
- السطر 32. هذه التعليمة تضع في المتغير `styleN` كائن من صنف **ParagraphStyle()** الذي يقوم بتعريف النمط الذي تم اختياره لكافة الفقرات، وفي هذه الحالة نمط النص الحالي بخط Times-Roman. كما سبق ذكره أعلاه، ويمكننا بسهولة تغيير نمط بتغيير القيمة الافتراضية من سماته المختلفة (انظر التمارين في نهاية الفصل) .
- الأسطر من 34 إلى 44 : هذا هو مكان الذي سنبني قائمة كائنات fluables التي سوف تتلقى المزيد، من التدفق المستمر، في مختلف الأطر التي أعددناها. وسوف تكون هذه القائمة نوع ما "قصة" وسوف نقوم بوضع أجزاء نصوص، وصور، إلخ، في أماكن مختلفة مخصصة لصفحاتنا. وهذا ما يفسر اختيار اسم المتغير `story` عادة للإشارة إلى هذه القائمة. لكل سطر مستخرج من الملف النصي، سوف نصيف إليه كائن fluable من نوع فقرة، تم تمثيله في السطر 38 مع نمط معرفة في `styleN` تليها مباشرة fluable من نوع مسافات، يتم إنشاء المثيل في السطر 40. بعد بضعة فقرات، سوف نستدعي أيضا بعض fluables من نوع صورة (الأسطر 42 و 43). هذه الصور يمكن إعادة تحجيمها حسب رغبتك (البرامتر الاختياري `kind="proportional"` لحفظ قوة نسبة جانب (و هذا يعني نسبة الطول/العرض، والأبعاد التي

تسبقها هي الحد الأقصى). وبالمناسبة، نحن قمنا بعد الفقرات المتوازية و fluables في المتغيرات **n** و **f**، لكن من الواضح أنه ليس ضروريا .

• الأسطر من 49 إلى 52 : في الصفحة الأولى من وثيقتنا، قمنا بتمثيل 3 أطر (كائنات من صنف **Frame()** لـ ReportLab) : عمودان عموديان فوق مستطيل أفقي. هذه الأطر هي من المساحات "التي تتقضى" fluables. لكل إطار، يجب توفير بالترتيب : إحداثيات الزاوية اليسرى السفلية للمستطيل (يبدأ الحساب من أسفل الصفحة)، وعرضه وارتفاعه. وليبقى الكود قابلا للقراءة، قمنا بالإشارة إلى الأبعاد بالسنتمتر، وقمنا بتحويلها إلى نقاط بيكا مع استخدام الثابت **cm**. البرامتر الاختياري **showBoundary=1** يسمح بتصور الإطارات المستطيلة بشكل فعال خلال مراحل تطوير سكريبتاتك. عند الانتهاء منها، يمكنك ببساطة حذف هذه الحجة (أو إعطاؤها قيمة 0) لجعلها تختفي.



Cela peut paraître paradoxal, mais comme nous l'avons déjà fait remarquer plus haut, le vrai maître est en fait celui qui ne croit à aucune magie, à aucun don, à aucune intervention surnaturelle.

Seule la froide, l'implacable, l'inconfortable logique est de mise.

Le mode de pensée d'un programmeur combine des constructions intellectuelles complexes, similaires à celles qu'accomplissent les mathématiciens, les ingénieurs et les scientifiques.

Comme le mathématicien, il utilise des langages formels pour décrire des raisonnements (ou algorithmes). Comme l'ingénieur, il conçoit des dispositifs, il assemble des composants pour réaliser des mécanismes et il évalue leurs performances. Comme le scientifique, il observe le comportement de systèmes complexes, il crée des modèles, il teste des prédictions.

L'activité essentielle d'un programmeur consiste à résoudre des problèmes.

Il s'agit-là d'une compétence de haut niveau, qui implique des capacités et des connaissances diverses : être capable de (re)formuler un problème de plusieurs manières différentes, être capable d'imaginer des solutions innovantes et efficaces, être capable d'exprimer ces solutions de manière claire et complète.

Comme nous l'avons déjà évoqué plus haut, il s'agira souvent de mettre en lumière les implications concrètes d'une représentation mentale « magique », simpliste ou trop abstraite.

La programmation d'un ordinateur consiste en effet à « expliquer » en détail à une machine ce qu'elle doit faire, en sachant d'emblée qu'elle ne peut pas véritablement « comprendre » un langage humain, mais seulement effectuer un traitement automatique sur des séquences de caractères.

Il s'agit la plupart du temps de convertir un souhait exprimé à l'origine en termes « magiques », en un vrai raisonnement parfaitement structuré et élucidé dans ses moindres détails, que l'on appelle un algorithme.

Considérons par exemple une suite de nombres fournis dans le désordre : 47, 19, 23, 15, 21, 36, 5, 12 …

Comment devons-nous nous y prendre pour obtenir d'un ordinateur qu'il les remette dans l'ordre ?

Fluable n° 77 Le souhait « magique » est de n'avoir qu'à cliquer sur un bouton, ou entrer une seule instruction au clavier, pour qu'automatiquement les nombres se mettent en place. Mais le travail du sorcier-programmeur est justement de créer cette « magie ».

Fluable n° 79 Pour y arriver, il devra décortiquer tout ce qu'implique pour nous une telle opération de tri (au fait, existe-t-il une méthode unique pour cela, ou bien y en a-t-il plusieurs ?), et en traduire toutes les étapes en une suite d'instructions simples, telles que par exemple « comparer les deux premiers nombres, les échanger s'ils ne sont pas dans l'ordre souhaité, recommencer avec le deuxième et le troisième, etc. ».

Fluable n° 82 Si les instructions ainsi mises en lumière sont suffisamment simples, il pourra alors les encoder dans la machine en respectant de manière très stricte un ensemble de conventions fixées à l'avance, que l'on appelle un langage informatique.

Fluable n° 84 Pour « comprendre » celui-ci, la machine sera pourvue d'un mécanisme qui décode ces instructions en associant à chaque « mot » du langage une action précise.

Fluable n° 86 Ainsi seulement, la magie pourra s'accomplir.

Fluable n° 88

• الأسطر من 54 إلى 56 : الأسلوب **addFromList()** هي كائنات إطارات تم صنعها بالخطوة السابقة تقوم بالملء، بداية من القائمة **story**. فهي تقوم باستخراج ال fluables واحدا واحدا وتثبيتها واحد تحت الأخر في الإطار، حتى يمتلئ، ومن ثم يقفز تلقائيا إلى السطر الضروري لتجنب انقسام الكلمات، ويطبق جميع مؤشرات الأنماط التي تم اختيارها. عندما يمتلئ الإطار، يمكن استخدام القائمة المتبقية في **story** لملئ أطر أخرى، وهكذا.

> لاحظ أنه إذا كانت الفقرة كبيرة جدا ليتم تثبيتها في الإطار، يتم صنع استثناء. والذي يمكن أن يتسبب في الكشف عن تقسيم تلقائي لفقرة في فقرات أصغر. راجع وثائق *Reportlab* لمزيد من المعلومات .

• السطر 58 يقوم بإغلاق الصفحة الحالية ويقوم بإدراج صفحة جديدة في المستند(مستند).

• الأسطر 60 إلى 64 : الصفحة المدرجة حديثا فارغة. لأنه يثبت أطر أخرى، ونملؤه بـ fluables التي نستمر باستخراجها من قائمة **story**ـ لكن في نهاية العملية لن تكون فارغة : سوف نقوم بخدمة الـ fluables المتبقية لشرح أكثر أساسية. والسماح بالتحكم أكثر لوضعهم في الصفحة .

• الأسطر 67 و 68 : سوف نختار نقطة بداية على الصفحة (العد يبدأ دائما من الأسفل !) وأبعاده الأقصىـ (العرض والإرتفاع) التي قررنا تخصيصها لكل من fluablesتنا. في هذه الحالة، إخترنا مربع من 14×14 سم. هتين القيمتين، العرض الفقط الذي سيتم حده، لأن أي من fluables المتبقية هي كبيرة بما يكفي لإرتفاع 14 سم.

• الأسطر من 69 إلى 75 : سوف نقوم بتدوير قائمة fluables المتبقية. بدلا من "التدفق" في أطر صارمة معرفة سابقا كما فعلنا حتى الآن، وهذه المرة سوف نقوم بوضعها مباشرة على اللوحة، بتحديد المساحة المطلوبة لكل واحدة. الأسلوب **wrap()** للكائن fluable أثناء معالجتها لهذا الغرض. ويأخذ برامترات الأبعاد القصوى للمساحة التي ترغب في تخصيصها لـ fluable (و هذا يعني بشكل عام، العرض الذي تريد رؤيته، والإرتفاع الواضح أكثر أهمية)، وتقوم بإرجاع العرض والإرتفاع التي سيتم استخدامها. في مثالنا اخترنا عرض وارتفاع ثابت، وجدنا أنه يمكننا وضع ال fluables تحت الأخريات (الأسلوب **drawOn()**)ـ السطر 74) بالحفاظ على معرفة الإحداثيات الفعلية في أي وقت (و هذ ما يسمح لنا على سبيل المثال وضع ملصق بجوار كل فقرة (السطر 73)). الأسلوب **identity()** يستخدم في السطر 72 والذي يسمح بالمناسبة بتحديد أي نوع (فقرة، صورة، فراغات) لـ fluable أثناء المعالجة .

في الختام

اعلم أن المذكور هنا هو لمحة صغيرة جدا من إمكانيات الموارد الضخمة التي تقدمها مكتبة ReportLab. من الواضح أننا تغاضينا عن الكثر من المفاهيم والأساليب. على سبيل المثال نحن لم نشرح ماذا نفعل لخفض فقرة كبيرة جدا أو إنشاء جداول أو رسوم بيانية وقوالب للمستندات، وصلات وما إلى ذلك ... ونحن لم نشرح حتى كيفية تشفير ال PDF الخاصة بنا أو أن

نضيف نماذج أو تأثيرات الإنتقال في الصفحة إلخ. مرة أخرى نحن نوصي بشدة أن وقم بالبحث على الإنترنت عن الوثائق المتوفرة لهذه المكتبة، فضلا عن مكتبة Python Imaging Library .

تمارين

1.18 قم بتعديل السكريبت السابق وذلك بإزالة الإطارات المستطيلة التي تركناها ليعرض أغراض العرض التوضيحي، وعدل نمط الفقرات المستخدم بتعديل سمات الكائن المناسب، مع مثالا :

```
styleN.fontName ='Helvetica-oblique'
styleN.fontSize =10
styleN.leading =11                    # تباعد
styleN.alignment =TA_JUSTIFY          # ou TA_LEFT، TA_CENTER أو TA_RIGHT
styleN.firstLineIndent =20            # بادئة للسطر الأول
styleN.textColor ='navy'
```

> لاحظ أن الثوابت *TA_CENTER*، *TA_RIGHT*، *TA_LEFT* و *TA_JUSTIFY* يمكن إستدعاؤها من وحدة *reportlab.lib.enums*. وهي على التوالي 0، 2، 1 و 4.

2.18 عدل السكريبت السابق لتقوم بتغييرات في النمط، على سبيل المثال، تقسم فقرة على ثلاثة. لفعل هذا، أنت تعرف أنه لا يمكنك نقل نمط تعيين بسيط مباشرة في متغير آخر، وذلك بسبب الأسماء المستعارة (انظر للصفحة 158، "نسخ قائمة"). لعمل نسخة "حقيقية" من كائن بيثون، ويمكنك استخدام الدالة **deepcopy()** باستدعاء الوحدة **copy** :

```
from copy import deepcopy
styleB = deepcopy(styleN)
```

3.18 عدل السكريبت السابق لطلب صفحة تحتوي على نص "مغلف" صورة. يمكن للصورة أن تكون بأي حجم، لكن سوف ترمز دائما تلقائيا على وسط العمود، ومحاذاة على الهامش الأيمن. حوله يتم تلقائيا تعريف ثلاثة أطر، ترتب تلقائيا الفقرات المختلفة التي أدرجت في هذه الأطر التي تجاوز تلقائيا الصورة.

Gestion des paragraphes avec ReportLab

La **programmation** est l'art d'apprendre à une machine comment accomplir de nouvelles tâches, qu'elle n'avait jamais été capable d'effectuer auparavant.
C'est par la programmation que vous pourrez acquérir le plus de contrôle, non seulement sur votre machine, mais aussi peut-être sur celles des autres par l'intermédiaire des réseaux. D'une certaine façon, cette activité peut donc être assimilée à une forme particulière de magie.

Elle donne effectivement à celui qui l'exerce un certain pouvoir, mystérieux pour le plus grand nombre, voire inquiétant quand on se rend compte qu'il peut être utilisé à des fins malhonnêtes.

Dans le monde de la programmation, on désigne par le terme ***hacker*** les programmeurs chevronnés qui ont perfectionné les systèmes d'exploitation de type Unix et mis au point les techniques de communication qui sont à la base du développement extraordinaire de l'Internet.
Ce sont eux également qui continuent inlassablement à produire et à améliorer les logiciels libres (*Open Source*).

Selon notre analogie, les hackers sont donc des maîtres-sorciers, qui pratiquent la magie blanche.

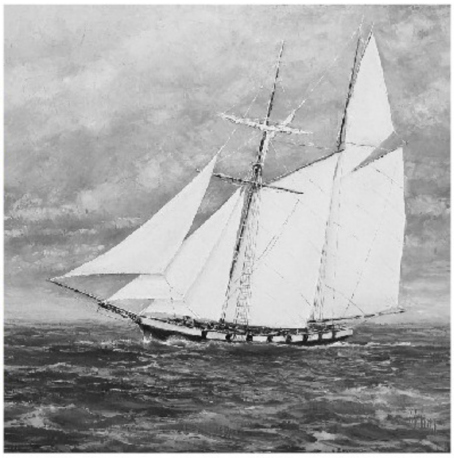
Mais il existe aussi un autre groupe de gens que les journalistes mal informés désignent erronément sous le nom de *hackers*, alors qu'ils devraient plutôt les appeler *crackers*.

Ces personnes se prétendent *hackers* parce qu'ils veulent faire croire qu'ils sont très compétents, alors qu'en général ils ne le sont guère.

Ils sont cependant très nuisibles, parce qu'ils utilisent leurs quelques connaissances pour rechercher les moindres failles des systèmes informatiques construits par d'autres, afin d'y effectuer toutes sortes d'opérations illicites : vol d'informations confidentielles, escroquerie, diffusion de spam, de virus, de propagande haineuse, de pornographie et de contrefaçons, destruction de sites web, etc.

Ces sorciers dépravés s'adonnent bien sûr à une forme grave de magie noire.
Mais il y en a une autre.

Les vrais hackers cherchent à promouvoir dans leur domaine une certaine éthique, basée principalement sur l'émulation et le partage des connaissances. La plupart d'entre eux sont des perfectionnistes, qui veillent non seulement à ce que leurs constructions logiques soient efficaces, mais aussi à ce qu'elles soient élégantes, avec une structure parfaitement lisible et documentée.



Vous découvrirez rapidement qu'il est aisé de produire à la va-vite des programmes qui fonctionnent, certes, mais qui sont obscurs et confus, indéchiffrables pour toute autre personne que leur auteur (et encore !).
Cette forme de programmation absconse et ingérable est souvent aussi qualifiée de « magie noire » par les *hackers*.

La démarche du programmeur

Comme le sorcier, le programmeur compétent semble doté d'un pouvoir étrange qui lui permet de transformer une machine en une autre, une machine à calculer en une machine à écrire ou à dessiner, par exemple, un peu à la manière d'un sorcier qui transformerait un prince charmant en grenouille, à l'aide de quelques incantations mystérieuses entrées au clavier.
Comme le sorcier, il est capable de guérir une application apparemment malade, ou de jeter des sorts à d'autres, via l'Internet.

Mais comment cela est-il possible ?

Cela peut paraître paradoxal, mais comme nous l'avons déjà fait remarquer plus haut, le vrai maître est en fait celui qui ne croit à aucune magie, à aucun don, à aucune intervention surnaturelle.
Seule la froide, l'implacable, l'inconfortable logique est de mise.

Le mode de pensée d'un programmeur combine des constructions intellectuelles complexes, similaires à celles qu'accomplissent les mathématiciens, les ingénieurs et les scientifiques.

Comme le mathématicien, il utilise des langages formels pour décrire des raisonnements (ou algorithmes). Comme l'ingénieur, il conçoit des dispositifs, il assemble des composants pour réaliser des mécanismes et il évalue leurs performances. Comme le scientifique, il observe le comportement de systèmes complexes, il crée des modèles, il teste des prédictions.

4.18 في سكريبت **spectacles.py** في الفصل السابق، أضف ما يلزم لجعل موضوع الإدارة تكون قائمة الحجوزات الـتي تقترحها الزائر بالحصول عليها على شكل وثيقة PDF :



Grand Théâtre de Python City

Les réservations ci-après ont déjà été effectuées :

Titre	Nom du client	Courriel	Places réservées
La souris qui en savait trop	Billou	bill@linuxfans.com	5
Le hacker et la princesse	Billou	bill@linuxfans.com	5
Le hacker et la princesse	Tux	tux@windoze.net	4
Zorro et les vampires	Tux	tux@windoze.net	3

Veuillez cliquer ici pour accéder au document PDF correspondant.

Retour à la page d'accueil



Grand théâtre de Python city

Titre	Nom du client	Courriel	Places réservées
La souris qui en savait trop	Billou	bill@linuxfans.com	5
Le hacker et la princesse	Billou	bill@linuxfans.com	5
Le hacker et la princesse	Tux	tux@windoze.net	4
Zorro et les vampires	Tux	tux@windoze.net	3

19

الاتصال عبر الشبكة وخاصية التعدد (multithreading)

أثبت التطور غير العادي للإنترنت بوضوح أن أجهزة الحاسوب يمكن أن تكون أدوات اتصال فعالة للغاية . في هذا الفصل، سوف نستكشف أساسيات هذه التكنولوجيا، عن طريق إجراء بعض التجارب مع أساليب بسيطة لربط الاتصال بين البرامج، لتسليط الضوء على ربط المعلومات بين العديد من الشركاء .

في ما يلي، نحن نفترض أنك تتعاون مع أشخاص أخرين، ومحطة عملك البايقون متصل بشبكة محلية باستخدام بروتوكول *TCP/IP* . إن نظام التشغيل غير مهم : على سبيل المثال، يمكنك تثبيت سكريبتات بيثون التي تم وصفها أدناه على محطة عمل على نظام تشغيل لينكس، والتواصل مع سكريبت أخر يتم تنفيذه على محطة عمل بنظام تشغيل مختلف، مثل ماك أو ويندوز .

يمكنك أيضا محاولة ما يلي على جهاز واحد، بوضع السكريبتات المختلفة في نوافذ منفصلة .

sockets ال

التمرين الأول الذي سوف يتم تقديمه هو إقامة اتصال بين جهازين فقط . يمكن لكل واحدة منهم تبادل الرسائل بالدور، لكن سوف تجد أن التكوينات ليست متماثلة . سيقوم السكريبت المثبت على واحد من هذه الأجهزة بدور برنامج الخادم، في حين أن الأخر سيكون بمثابة برنامج العميل .

برنامج الخادم يعمل بشكل مستمر على جهاز هويته معرفة جيدا على الشبكة عن طريق عنوان IP معين[106] . وهو ينتظر وصول الطلبات المرسلة من قبل العملاء المحتملين لهذا العنوان، من خلال خاصية منفذ الاتصالات المحدد . للقيام بذلك، يجب على السكريبت أن ينفذ كائنا برمجيا مرتبطا بهذا المنفذ والذي يسمى socket .

و من خلال جهاز آخر، يحاول برنامج العميل الاتصال عن طريق طلب مناسب . هذا الطلب هو رسالة ترسل للشبكة، وتماما كما تكلف الرسالة إلى مكتب البريد . يمكن للشبكة توجيه الطلب إلى أي آلة أخرى. ولكن بشرط : ليمكن الوصول إلى الجهة المقصودة، يجب أن يحتوي الطلب في رأيه إشارة إلى عنوان IP ومنفذ الاتصال للمتلقين .

عندما يتم إنشاء اتصال مع الخادم، يعين العميل بنفسه أحد منافذه للاتصالات الخاصة . ومن هذه اللحظة، يمكننا أن نعتبر هذه قناة متميزة تربط بين جهازين، كما لو كانت متصلة مع بعضها البعض عن طريق الأسلاك(منفذا الاتصال يعملان دور طرفي السلك) . ويمكن بدأ تبادل المعلومات .

لاستخدام منافذ اتصال الشبكة، تقوم البرامج باستدعاء مجموعة من إجراءات ودالات نظام التشغيل، من خلال كائنات الواجهة تسمى sockets . وهذه يمكن أن تنفذ تقنيتي اتصالات مختلفتين ومتكاملتين : وهي الحزم (وتسمى أيضا حزم البيانات)، تستخدم على نطاق واسع على شبكة الإنترنت، ويستمر الاتصال، أو تدفق socket، والذي هو أبسط قليلا .

بناء خادم بدائي

لتجاربنا الأولى، سوف نستخدم تقنية تدفق ال sockets .

هذا في الواقع مناسب جدا عندما يتعلق الأمر بالتواصل بين أجهزة الحاسوب المتصلة عبر شبكة محلية . وهذا الأسلوب تنفيذه سهل للغاية، فهو يسمح بإنتاجية عالية لتبادل البيانات .

و أما عن التقنية الأخرى (الحزم) من الأفضل أن تستخدم للاتصالات المرسلة عبر شبكة الإنترنت، وذلك بسبب موثوقيتها العالية(الحزم نفسها يمكن أن تصل إلى وجهتها من خلال مسارات مختلفة، أو أن تصدر أو تعيد إصدار نسخ متعددة إذا كان ذلك ضروريا لتصحيح أخطاء الإرسال)، لكن تنفيذها أكثر تعقيدا نوعا ما . ونحن لن ندرسها في هذا الكتاب .

السكريبت الأول أدناه هو خادم قادر على التواصل مع عميل واحد . سوف نرى لاحقا ماذا يجب علينا أن نضيف حتى نتمكن من دعم الاتصالات المتوازية من عدة عملاء .

```
1# # تعريف خادم ويب بدائي
2# # هذا الخادم ينتظر اتصال عميل
3#
4# import socket, sys
5#
```

[106] و يمكن لجهاز خاص أن يقوم بتعيينه بأكثر وضوح. و لكن شرط أن يتم وضع آلية على شبكة (DNS) لترجمة عنوان ال IP تلقائيا . يرجى الرجوع إلى كتاب عن الشبكات لمزيد من المعلومات .

```
 6# HOST = '192.168.1.168'
 7# PORT = 50000
 8# counter =0          # عداد الاتصالات النشطة
 9#
10# # 1) صنع socket :
11# mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
12#
13# # 2) ربط socket إلى عنوان محدد :
14# try:
15#     mySocket.bind((HOST, PORT))
16# except socket.error:
17#     print("La liaison du socket à l'adresse choisie a échoué.")
18#     sys.exit
19#
20# while 1:
21#     # 3) انتظار استعلام اتصال لعميل :
22#     print("Serveur prêt, en attente de requêtes ...")
23#     mySocket.listen(2)
24#
25#     # 4) إجراء الاتصال :
26#     connexion, adresse = mySocket.accept()
27#     counter +=1
28#     print("Client connecté, adresse IP %s, port %s" % (adresse[0], adresse[1]))
29#
30#     # 5) التحاور مع العميل :
31#     msgServeur ="Vous êtes connecté au serveur Marcel. Envoyez vos messages."
32#     connexion.send(msgServeur.encode("Utf8"))
33#     msgClient = connexion.recv(1024).decode("Utf8")
34#     while 1:
35#         print("C>", msgClient)
36#         if msgClient.upper() == "FIN" or msgClient =="":
37#             break
38#         msgServeur = input("S> ")
39#         connexion.send(msgServeur.encode("Utf8"))
40#         msgClient = connexion.recv(1024).decode("Utf8")
41#
42#     # 6) إغلاق الاتصال :
43#     connexion.send("fin".encode("Utf8"))
44#     print("Connexion interrompue.")
45#     connexion.close()
46#
47#     ch = input("<R>ecommencer <T>erminer ? ")
48#     if ch.upper() =='T':
49#         break
```

تعليقات

- السطر 4 : وحدة **socket** تحتوي على جميع دالات وأصناف اللازمة لبناء برنامج تواصل . كما سنرى في السطور التالية، وإنشاء الاتصال يشمل 6 خطوات .
- السطران 6 و 7 : هذان المتغيران يعرفان هوية الخادم، لدمجه مع socket . يحتوي **HOST** على سلسلة نصية تشير إلى عنوان IP للخادم في شكل عشري معتاد، أو اسم DNS لنفس الخادم (ولكن بشرط أن يتم تنفيذ آلية تحليل الاسم

على الشبكة) . ويجب على المتغير **PORT** أن يحتوي على عدد صحيح، أي رقم منفذ لا يستخدم لغرض آخر، ويفضل أن تعطي قيمة أكبر من 1024 ..

- السطور من 10 إلى 11 : الخطوة الأولى لآلية الربط البيني . ولقد قمنا بتمثيل كائن من صنف **socket()**, ونحدد خيارين للإشارة إلى نوع العنوان الذي تم اختياره (نحن استخدمنا عناوين من نوع "Internet") فضلا عن تكنولوجيا النقل (حزم بيانات أو اتصال مستمر (تدفق) : قررنا استخدام هذا الأخير).
- السطور من 13 إلى 18 : الخطوة الثانية . نحاول تأسيس اتصال بين socket ومنفذ الاتصال . إذا لا يمكن إنشاء اتصال (على سبيل المثال، إذا كان منفذ الاتصال مشغولا أو اسم الجهاز خاطئ)، يتم إغلاق البرنامج مع رسالة خطأ . في السطر 15، لاحظ أن الأسلوب **bind()** ل socket ينتظر برامترا من نوع مصفوفة مغلقة، لذا يجب علينا أن نحصر متغيراتنا في زوج من الأقواس المزدوجة .
- السطر 20 : تم تصميم برنامج الخادم الخاص بنا ليعمل باستمرار في انتظار طلبات العملاء المحتملين، لقد وضعنا حلقة لانهائية .
- السطور من 21 إلى 23 : الخطوة الثالثة . سيتم ربط socket مع منفذ الاتصال، ويمكن الآن أن يكون على استعداد لتلقي الطلبات المرسلة من قبل العملاء . وهذا هو دور الأسلوب **listen()**. البرامتر الذي ينقل يشير إلى العدد الأقصى من الاتصالات التي يجب عليه قبولها في وقت لاحق . وسوف نرى لاحقا كيفية إدارتها .
- السطور من 25 إلى 28 : الخطوة الرابعة . عندما نقوم باستدعاء الأسلوب **accept()**, ينتظر socket إلى مالانهاية أحد الطلبات . ويتم قطع السكريبت في هذه المرحلة، تماما مثل عندما نستدعي الدالة **input()** لانتظار إدخال من لوحة المفاتيح . فإذا تم استلام طلب، الأسلوب **accept()** يقوم بإرجاع نفق من عنصرين : الأول هو مرجع كائن جديد للصنف **socket()**[107], والذي سيكون واجهة الاتصال الحقيقي بين الخادم والعميل، والثاني مصفوفة مغلقة أخرى تحتوي على إحداثيات هذا العميل (عنوان الـ IP الخاص بم ورقم المنفذ الذي يستخدمه) .
- السطور من 30 إلى 33 : الخطوة الخامسة . يتم تأسيس الاتصال الفعلي . الأساليب **send()** و **recv()** لـ socket والتي تقوم بنقل واستقبال الرسائل . والتي يجب أن تكون سلاسل بيتات . عند الإرسال، فمن الضروري أن يتم إدراج سلاسل نصية من البيانات من نوع byte، وتقوم بالعكس عند الاستقبال .

107سوف نرى لاحقا الفائدة من إنشاء كائن جديد socket ليتولى أمر الاتصالات، الذي قام المستخدم بإنشائه في السطر 10. و باختصار، إذا أردنا أن يقوم خادمنا بدعم إتصالات من أكثر من عميل في نفس الوقت، يجب أن نقوم بعمل socket مستقل لكل منهم، و يكون مستقل عن الأول الذي سيقوم بشكل مستمر بتلقي الاستعلامات من العملاء الجدد.

> يقوم الأسلوب **send()** بإرجاع عدد البايتات المرسلة . واستدعاء الأسلوب **recv()** يجب أن يحتوي على برامتر عدد صحيح يشير إلى العدد الأقصى من وحدات البايت التي يتم إرسالها دفعة واحدة . والبايتات الفائضة يتم وضعها في مخزن مؤقت، ويتم إرسالها عندما يتم استدعاء الأسلوب **recv()** من جديد.

- السطور من 34 إلى 40 : هذه الحلقة الجديدة لانهائية حتى تحافظ على التواصل إذا قرر العميل إرسال كلمة "fin" أو سلسلة بسيطة فارغة . شاشات عرض للجهازين تقوم بعرض كل تطور في هذا الحوار .
- السطور من 42 إلى 45 : الخطوة السادسة . إغلاق الاتصال .

بناء عميل بدائي

السكريبت بالأسفل يعرف برنامج عميل متكامل مع الخادم الذي تم شرحه في الصفحات السابقة . لقد قمنا بتبسيطه .

```
# تعريف عميل شبكة بدائي
# هذا العميل يتحاور مع خادم مخصص

import socket, sys

HOST = '192.168.1.168'
PORT = 50000

# 1) صنع socket :
mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# 2) إرسال إستعلام اتصال إلى الخادم :
try:
    mySocket.connect((HOST, PORT))
except socket.error:
    print("La connexion a échoué.")
    sys.exit()
print("Connexion établie avec le serveur.")

# 3) التحاور مع الخادم :
msgServeur = mySocket.recv(1024).decode("Utf8")

while 1:
    if msgServeur.upper() == "FIN" or msgServeur =="":
        break
    print("S>", msgServeur)
    msgClient = input("C> ")
    mySocket.send(msgClient.encode("Utf8"))
    msgServeur = mySocket.recv(1024).decode("Utf8")

# 4) إغلاق الاتصال :
print("Connexion interrompue.")
mySocket.close()
```

تعليقات

- بداية هذا السكريبت تشبه الخادم . يجب على عنوان ال IP والمنفذ أن يكونا مطابقين للخادم .

- * السطور من 12 إلى 18 : نحن لا نصنع هذه المرة كائن socket واحدا، والذي يستخدم الأسلوب **connect()** لإرسال طلب اتصال .
- السطور من 20 إلى 33 : بمجرد تأسيس الاتصال، يمكنك التواصل مع خادم يستخدم الأساليب **send()** و **recv()** التي سبق وتم شرحها أعلاه.

إدارة مهام متعددة في نفس الوقت باستخدام المواضيع (theards)

نظام الاتصالات الذي قمنا بتطويره في الصفحات السابقة بدائي جدا : فإنه من ناحية يربط جهازين، ومن ناحية أخرى يحد من حرية التعبير عن متحاورين . هذه في الحقيقة لا يمكنها إرسال رسائل بالدور . على سبيل المثال، عندما يرسل أحدهم رسالة، يقوم النظام بمنعه لأن شريكه الأخر لم يرسل رد . عندما يتعلق الأمر بتلقي الرد، يبقى النظام غير قادر على الاستقبال الأخر، لأنه لم يدخل بنفسه رسالة جديدة وهكذا ...

كل هذه المشاكل تنبع من حقيقة أن سكريبتاتنا معتادة على التعامل مع شيء واحد فقط في كل مرة . على سبيل المثال، عندما يواجه تدفق التعليمات الدالة **input()**، لا يحدث أي شيء حتى يقوم المستخدم بإدخال البيانات المتوقعة، وحتى لو كان هذا يأخذ وقت طويلا جدا، فإن من العادة يكون من غير الممكن أن يقوم البرنامج بتنفيذ مهام أخرى في هذا الوقت . ومع ذلك، هذا صحيح فقط في نفس البرنامج الواحد : ربما كنت تعلم أنه يمكنك تشغل تطبيقات أخرى على حاسوبك لأن أنظمة التشغيل الحديثة متعددة المهام .

الصفحات التالية سوف نشرح بها كيف يمكنك تقديم ميزة تعدد المهام في برامجك، حتى تتمكن من تطوير تطبيقات شبكة حقيقية، يمكنها الاتصال في وقت واحد مع عدة شركاء .

يرجى الآن النظر إلى السكريبت في الصفحة السابقة . وتتمثل ميزته الرئيسية في حلقة **while** في السطور من 23 إلى 29 . ومع ذلك، يتم مقاطعة هذه الحلقة في مكانين

- في السطر 27، في انتظار مدخلات من لوحة المفاتيح من المستخدم (الدالة **input()**) ;
- في السطر 29، في انتظار وصول رسالة الشبكة.

هذان المتوقعان متعاقبان، فإنه سيكون أكثر إثارة للاهتمام إذا كانا في الوقت نفسه . إذا كانت هذه هي الحالة، يمكن للمستخدم إرسال رسائل في أي وقت، دون الحاجة إلى انتظار في كل مرة ردة فعل الشريك . وقد يظهر أي عدد من الرسائل، دون وجود إلزام للرد على كل واحدة منها لاستقبال الأخريات .

يمكننا تحقيق هذا إذا تعلمنا إدارة المتسلسلات المتعددة للتعليمات بالتوازي ضمن برنامج واحد . ولكن كيف يمكن أن يكون هذا ممكنا ؟

على مدى تاريخ الحاسوب، وضعت العديد من التقنيات لتقاسم وقت عمل المعالج بين المهام المختلفة، بحيث تبدو وكأنها صنعت في نفس الوقت (في حين أن المعالج سيعطي لك دورة واحدة) . يتم تطبيق هذه التقنية في نظام التشغيل، وأنه ليس ضروري أن نقوم بشرحها بالتفصيل هنا، على الرغم من أننا نستطيع الوصول إلى كل واحدة منهم عن طريق بيثون .

في الصفحات التالية، سوف تتعلم كيفية استخدام واحدة من هذه التقنيات فقط وهي سهلة ومحمولة حقا (و هي في الحقيقة تدعم أنظمة التشغيل الرئيسية) ونسمي هذه التقنية العمليات الخفيفة أو الخيوط threads[108].

في برنامج الحاسوب، المواضيع هي تيارات من التعليمات التي تنفذ بالتوازي (في وقت واحد تقريبا)، في حين تتشارك في نفس مساحة الأسماء العامة .

في الحقيقة، إن أي تدفق عمليات لأي برنامج بيثون سوف يحتوي على الأقل على موضوع : الموضوع الرئيسي . ومن هذا، يمكن لمواضيع الأطفال أن يبدؤوا، وسيتم تنفيذها في نفس الوقت . كل موضوع طفل ينتهي ويختفي دون مزيد من اللغط عندما يتم تنفيذ كافة تعليماته . وعندما ينتهي الموضوع الرئيسي، فمن الضروري في بعض الأحيان ضمان أن جميع مواضيعه الأطفال قد تم "قتلهم" معه .

عميل شبكة لإدارة الإرسال والاستقبال المتزامن

سوف نطبق الآن تقنية المواضيع لبناء نظام دردشة[109] مبسط . هذا النظام يتكون من خادم واحد وأي عدد من العملاء . على عكس ما حدث في تمريننا الأول، لا يستخدم أحد خادم نفسه للاتصال، لكن عندما يتم تشغيلها، يمكن للعديد من العملاء الاتصال به والبدء في تبادل الرسائل .

كل عميل سيقوم بإرسال كافة الرسائل إلى الخادم، ولكن سوف يتم إحالة الرسائل على الفور على جميع العملاء الأخرين المتصلين، بحيث رؤية كل واحد حركة المرور . يستطيع كل واحد منهم إرسال رسائل ويتلقاها الأخرون في أي وقت، في أي ترتيب، والتلقي والإرسال سوف تدار في مواضيع منفصلة .

السكريبت النصي أدناه يقوم بتعريف برنامج عملي . وسيتم شرح برنامج الخادم لاحقا . وسوف تجد أن الجزء الرئيسي من البرنامج النصي (السطر 38 وما يليه) مشابه للمثال السابق . فقط جزء "الحوار مع الخادم" سوف يتم استبداله . بدلا من حلقة **while**, ستجد الآن تعليمات لصنع كائنين (في السطرين 49 و 50)، لنبدأ مميزات هذين السطرين التاليين . هذان الكائنان موضوعان تم صنعهما عن طريق اشتقاق من صنف **Thread()** من وحدة **threading**. الذين يشغلونها بغض النظر عن استقبال وإرسال الرسائل . ويتم تغليف الموضوعين الطفلين في كائنات منفصلة، مما يسهل فهم الآلية

108 في نظام تشغيل من نوع يونكس (مثل لينكس)، الخيوط المختلفة لبرنامج نفسه هي جزء من عملية واحدة . و من الممكن أيضا إدارة العمليات المختلفة باستخدام سكريبت بيثون (عملية متفرقة)، و لكن تفسير هذه التقنية خارج نطاق هذا الكتاب .

109 "الشات" (باللغة الفرنسية) هي عملية دردشة عبر أجهزة الحاسوب. و لقد تم إقتراح هذا المصطلح من قبل فرانكفونيين كنديين و يشير إلى "الدردشة عبر لوحة المفاتيح" .

```
# تعرف عميل مدير شبكة يعمل بالتوازي في نقل واستقبال الرسائل (باستخدام خيطين) .


host = '192.168.1.168'
port = 46000

import socket, sys, threading

class ThreadReception(threading.Thread):
    """objet thread gérant la réception des messages"""
    def __init__(self, conn):
        threading.Thread.__init__(self)
        self.connexion = conn           # مرجع socket الاتصال

    def run(self):
        while 1:
            message_recu = self.connexion.recv(1024).decode("Utf8")
            print("*" + message_recu + "*")
            if not message_recu or message_recu.upper() =="FIN":
                break
        # ينتهي هنا خيط <réception>.
        # نفرض إغلاق الخيط <émission> :
        th_E._stop()
        print("Client arrêté. Connexion interrompue.")
        self.connexion.close()

class ThreadEmission(threading.Thread):
    """objet thread gérant l'émission des messages"""
    def __init__(self, conn):
        threading.Thread.__init__(self)
        self.connexion = conn           # مرجع socket للاتصال

    def run(self):
        while 1:
            message_emis = input()
            self.connexion.send(message_emis.encode("Utf8"))

# البرنامج الرئيسي - تأسيس الاتصال :
connexion = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
    connexion.connect((host, port))
except socket.error:
    print("La connexion a échoué.")
    sys.exit()
print("Connexion établie avec le serveur.")

# الحوار مع الخادم : بتشغيل خيطين لإدارة مستقلة لعمليات إرسال وإستقبال الرسائل

th_E = ThreadEmission(connexion)
th_R = ThreadReception(connexion)
th_E.start()
th_R.start()
```

تعليقات

- ملاحظة هامة : في هذا المثال، قررنا إنشاء كائنين موضوعين مستقلين في الموضوع الرئيسي لتسليط الضوء بوضوح على الآليات . يستخدم برنامجنا 3 مواضيع في كل شيء، في حين أن القارئ اليقظ يرى أن اثنين تكفي . في الواقع، إن

الموضوع الرئيسي هو في نهاية المطاف مجرد مشغل 2 الأخريين ! لكن لا يوجد الحد الأدنى من عدد المواضيع . على العكس، منذ اللحظة التي قررت فيها استخدام هذه التقنية، يجب أن تأخذ ميزة لتقسيم التطبيق إلى وحدات متميزة .

- السطر 7 : تحتوي وحدة **threading** على تعريف مجموعة متنوعة من الأصناف المثيرة للاهتمام لإدارة المواضيع . سوف نستخدم هنا الموضوع الصنف الوحيد **Thread**()، لكن في وقت لاحق سوف نستغل (الصنف **Lock**())، عندما يكون لدينا ما يدعو للقلق حول قضايا التزامن بين المواضيع المتزامنة المختلفة .
- الأسطر من 9 إلى 25 : الصنف المشتق من صنف **Thread**() يحتوي على أساسيات الأسلوب **run**(). وهو في هذا المكان الذي هو جزء من برنامج عمل خصيصا في الموضوع . غالبا ما سيكون حلقة متكررة، مثل هنا . يمكنك أن تنظر بشكل كامل محتويات هذا الأسلوب كسكريبت مستقل، يعمل بالتوازي مع المكونات الأخرى من تطبيقك . عندما يتم تنفيذ هذا الكود تماما، يتم إغلاق الموضوع .
- الأسطر من 16 إلى 20 : تدير هذه الحلقة استقبال الرسائل . في كل تكرار، تدفق التعليمات يتوقف عند السطر 17 في انتظار رسالة جديدة، لكن لم يتم تجميد بقية البرنامج حتى الآن : مواضيع الأخرى تحتوي على عملهم بشكل المستقل .
- السطر 19 : تسبب الرسالة `'fin'` (كبيرة أو صغيرة) أو رسالة فارغة(هذا صحيح لاسيما إذا تم قطع الاتصال من قبل شريك) بخروج حلقة التلقي . فيتم تنفيذ بعض تعليمات "التنظيف"، وينتهي الموضوع .
- السطر 23 : عندما يتم إنهاء تلقي الرسائل، نأمل أن يتم إنهاء بقية البرنامج أيضا . لذلك نحن بحاجة إلى فرض إغلاق الكائنات المواضيع الأخرى، ونحن وضعناها في مكان مناسب لإدارة نقل الرسائل . ويمكن الحصول على هذا الإغلاق الإجباري باستخدام الأسلوب **_stop**()[110].
- الأسطر من 27 إلى 36 : هذا الصنف يعرف كائن موضوع آخر، الذي يحتوي هذه المرة على حلقة تكرار أبدية . لذا لا يمكن أن تنتهي إلا إذا أجبرنا إغلاقها من الأسلوب الذي تم شرحه في الفقرة السابقة . في كل حلقة من هذا التكرار، تدفق التعليمات يتوقف في السطر 35 في انتظار إدخال مدخلات من لوحة المفاتيح، ولكن هذا لا يمنع بأي حال من الأحوال المواضيع الأخرى من القيام بعملهم .
- السطور من 38 إلى 45 : هذه الأسطر تسرد مماثلة للسكريبتات السابقة .
- * السطور من 47 إلى 52 : يتم هنا تمثيل وبدء كائني موضوعين أطفال . يرجى ملاحظة أنه من المستحسن أن تبدأ هذه النتيجة من خلال استدعاء الأسلوب المدمج **start**()، بدلا من استدعاء الأسلوب **run**() مباشرة والذي قمت بتعريفه

110أرجو أن يسامحنا المبرمجون المتمرسون ؛ و أنا أعترف أن هذه الحيلة لفرض وقف للخيط ليست مستحسنة . ولكن وضعت هذا الاختصار لتجنب إثقال هذا النص. و هو مقدمة بدائية فقط . و يمكن للقارئ أن يستكشف هذه المسألة من خلال البحث في الكتب المرجعية المدرجة في قائمة المراجع .

بنفسك . لاحظ أيضا أنه يمكنك استدعاء **start()** مرة واحدة فقط (إذا توقف مرة، لن تستطيع إعادة تشغيل كائن الموضوع) .

خادم مدير شبكة اتصالات متعدد للعملاء في نفس الوقت

السكريبت النصي التالي يقوم بصنع خادم قادر على التعامل مع اتصالات لعدد من العملاء من نفس النوع الذي شرحناه في الصفحات السابقة .

لا يستخدم هذا الخادم للاتصال بنفسه : فالعملاء الذين سيتواصلون مع بعضهم البعض من خلال الخادم . فهي تلعب دور التتابع : فهو يقبل اتصالات العملاء، ثم ينتظر وصول رسائلهم . عندما تصل رسالة من عميل معين سيقوم الخادم بإعادة توجيهها إلى العملاء الآخرين (يمكننا أن نضيف إليه سلسلة تحديد الإرسال إلى عميل محدد)، بحيث يمكن للجميع رؤية كافة الرسائل، ومعرفة من أين جائت .

```
# تعريف خادم شبكة يدير نظام دردشة مبسط.
# استخدام الخيوط لصنع اتصالات عميل بالتوازي.

HOST = '192.168.1.168'
PORT = 46000

import socket, sys, threading

class ThreadClient(threading.Thread):
    '''dérivation d'un objet thread pour gérer la connexion avec un client'''
    def __init__(self, conn):
        threading.Thread.__init__(self)
        self.connexion = conn

    def run(self):
        # التحاور مع العميل :
        nom = self.getName()            # لكل خيط اسم
        while 1:
            msgClient = self.connexion.recv(1024).decode("Utf8")
            if not msgClient or msgClient.upper() =="FIN":
                break
            message = "%s> %s" % (nom, msgClient)
            print(message)
            # إرسال رسالة إلى جميع العملاء :
            for cle in conn_client:
                if cle != nom:          # لا يتم إعادة إرسالها إلى المرسل
                    conn_client[cle].send(message.encode("Utf8"))

        # إغلاق الاتصال :
        self.connexion.close()      # قطع الاتصال من جانب الخادم
        del conn_client[nom]        # حذف المدخلات في القاموس
        print("Client %s déconnecté." % nom)
        # ينتهي الخيط هنا

# تهيئة الخادم - وضع socket :
mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
    mySocket.bind((HOST, PORT))
```

```
39# except socket.error:
40#     print("La liaison du socket à l'adresse choisie a échoué.")
41#     sys.exit()
42# print("Serveur prêt, en attente de requêtes ...")
43# mySocket.listen(5)
44#
45# # انتظار ودعم الاتصالات المطلوبة من العملاء :
46# conn_client = {}                 # قاموس اتصالات العملاء
47# while 1:
48#     connexion, adresse = mySocket.accept()
49#     # صنع كائن خيط جديد لتوليد الاتصال :
50#     th = ThreadClient(connexion)
51#     th.start()
52#     # حفظ الاتصال في قاموس :
53#     it = th.getName()          # رقم الخيط
54#     conn_client[it] = connexion
55#     print("Client %s connecté, adresse IP %s, port %s." %\
56#            (it, adresse[0], adresse[1]))
57#     # التحاور مع العميل :
58#     msg ="Vous êtes connecté. Envoyez vos messages."
59#     connexion.send(msg.encode("Utf8"))
```

تعليقات

- السطور من 35 إلى 43 : تهيئة الخادم هي تعريف نفسه للخادم البدائي الذي تم شرحه في بداية الفصل السابق .
- * السطر 46 : مراجع المختلفة للاتصالات يجب أن يتم تخزينهم . يمكننا وضعها في قائمة، لكن من الأفضل وضعها في قاموس، وذلك لسببين : الأول هو أننا بحاجة إلى إضافة أو إزالة هذه المراجع في أي ترتيب، لأن العملاء سيتصلون ويقطعون الاتصال بإرادتهم . والسبب الثاني هو أننا يمكن أن نقوم بسهولة بتعريف معرف فريد لكل اتصال، والذي يمكن أن يكون بمثابة مفتاح وصول في القاموس . هذا المعرف سوف يقدم تلقائيا من قبل الصنف **Thread()**.
- السطور من 47 إلى 51 : يبدأ البرنامج هنا من خلال تكرار حلقة دائمة، والتي من شأنها أن تنتظر باستمرار الاتصالات الجديدة . لكل واحدة من هذه الاتصالات، يتم إنشاء كائن جديد **ThreadClient()** والتي يمكن العناية بها بشكل مستقل عن الآخرين .
- * السطور من 52 إلى 54 : الحصول على معرف فريد يتم باستخدام الأسلوب **getName()**. يمكننا هنا أن نتمتع لأن بيثون يقوم تلقائيا بتعيين اسم فريد لكل موضوع جديد : هذا الاسم كمعرف (أو مفتاح) للعثور على اتصال مناظر في قاموسنا . سوف ترى أنه عبارة عن سلسلة، من نموذج `"Thread-N"` (الحرف N يدل على ترتيب الموضوع) .
- السطور من 15 إلى 17 : ضع في اعتبارك أن الكائنات **ThreadClient()** كاتصال، وجميع هذه الكائنات تعمل بشكل مواز . الأسلوب **getName()** يمكن استخدامه داخل أي من هذه العناصر للعثور على هويته . ونحن نستخدم هذه المعلومات للتمييز بين الاتصال الحالي من جميع الآخرين (انظر للسطر 26) .

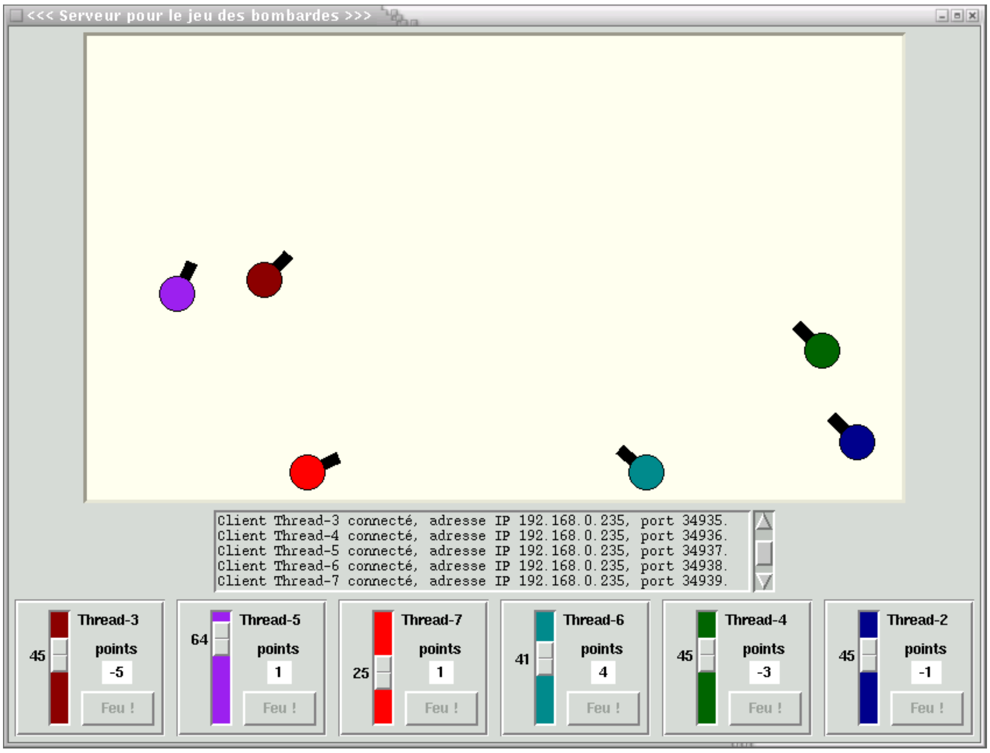
• السطور من 18 إلى 23 : فائدة الموضوع هو الحصول على كل الرسائل من عميل معين . لـذلك هـي حلقة دائمة التكرار، والتي سوف تتوقف عندما تستلم رسالة معينة "fin"، أو عند استلام رسالة فارغة (عند قطع الاتصال من شريك) .

• السطور من 24 إلى 27 : كل رسالة مستقبلة من عميل يجب إعادة توجيهها إلى كل الآخرين . نحن نستخدم هما حلقة **for** لتكرار على(التدوير) مفاتيح القاموس والاتصالات، والتي تسمح لنا بعد ذلك بإيجاد الاتصالات نفسها . وهنالك اختبار بسيط (السطر 26) يمنعنا من إعادة إرسال الرسالة إلى العميل الذي من أين يأتي .

• السطر 31 : عندما نغلق socket اتصال، فمن الأفضل إزالة مرجعها في القاموس، لأنه هذا المرجع لم يعد يستخدم . ويمكننا القيام بذلك دون اتخاذ احتياطات خاصة، حيث أن عناصر القاموس غير مرتبة (ويمكننا إضافة أو إزالة في أي ترتيب) .

لعبة القصف، نسخة الشبكة

في الفصل 15 علقنا في تطوير لعبة قتال صغيرة تواجه اللاعبين بالقصف . إن فائدة هذه اللعبة لا تزال محدودة للغاية، كما أنها تلعب على حاسوب واحد . سوف نقوم الآن، بإضافة التقنيات التي تعلمناها الآن . مثل نظام "الدردشة" الذي شرحناه في



الصفحات السابقة، فإن التطبيق الكامل سوف يتكون الآن من برنامجين متفصلين : برنامج خادم سيتم تشغيله على جهاز

واحد، وتطبيق العميل الذي يمكن تشغيله من أي جهاز .و نظر لطبيعة بيثون المحمولة، سوف تكون قادرا على تنظيم معارك بين أجهزة الحاسوب التي تديرها أنظمة تشغيل مختلفة (لينكس ↔ ويندوز ↔ ماك) .

برنامج الخادم : فكرة عامة

برامج الخادم والعميل يشغلون نفس قاعدة البرنامج، في حد ذاتها تعافى إلى حد كبير عن ماسبق التي وضعت في جميع أنحاء الفصل 15 . لذا نفترض هذا لما تبقى من هذه الدراسة التي تم حفظ نسختين سابقا من اللعبة في ملفات وحدات **canon03.py** و **canon04.py**, المثبتة في الدليل الحالي . في الواقع يمكننا استخدام الكثير من التعليمات البرمجية التي تحتويها، وسنستخدم بحكم استدعاء ووراثة الأصناف .

من وحدة **canon04**, سوف نعيد استخدام الصنف **Canon()** على هذا النحو، فإنه لكل من البرنامج الخادم للبرنامج العميل . من هذه الوحدة، سوف نستدعي الصنف **AppBombardes()**, والذي سوف نشتق منه الصنف الرئيسي من تطبيق خادمنا: **AppServeur()**. سوف تجد أيضا أنها تنتج بنفسها الصنف الفرعي **AppClient()**, ودائما عن طريق الميراث.

من وحدة **canon03**, سوف نستدعي صنف **Pupitre()** التي من شأنها أن تؤدي إلى نسخة أكثر ملائمة ل "التحكم عن بعد" .

و أخيرا، سيتم إضافة صنفين جديدين إلى ما سبق، كل واحدة متخصصة في إنشاء كائن موضوع : الصنف **ThreadClients()**, منها مثيل الذي يراقب باستمرار لتلقي طلبات socket من العملاء الجدد والصنف **ThreadConnexion()**, الذي سينشئ كائنات socket اللازمة لإجراء حوار مع كل عميل متصل بالفعل .

هذه الأصناف الجديدة ستكون مصدر إلهام من تلك التي قمنا بتطويرها لخادم الدردشة في الصفحات السابقة . والفرق الرئيسي هو أن لدينا ترابط محدد لتفعيل مواضيع خاصة للكود الذي ينتظر اتصالات العملاء، بحيث يمكن للتطبيق الرئيسي أن يفعل شيئا أخر خلال ذلك الوقت .

من هنا، عملنا الكبير الذي نواجه هو تطوير بروتوكول اتصال للحوار بين الخادم والعملاء . ما هو الجديد ؟ ببساطة نقوم بتعريف مضمون الرسائل التي سيتم تبادلها بين الآلات المتصلة . لا تقلق : تطوير هذه "اللغة" يمكن أن يكون متقدما . نبدأ من خلال إنشاء قاعدة حوار، ثم نضيف تدريجيا "مفردات" .

و يمكن تحقيق الكثير من هذا العمل من خلال مساعدة برنامج العميل الذي سبق ووضعناه لنظام الدردشة . ويستخدم لإرسال "الأوامر" إلى الخادم الذي نطوره، ويتم تصحيح الإطاعة : بوضوح، الإجراءات الذي سوف نضعها تدريجيا على خادم سوف يتم اختبارها تدريجبا، استجابة للرسائل التي أصدرها "باليد" العميل .

بروتوكول الاتصالات

و غني عن القول أن البروتوكول هو الموضح أدناه هو إجراء تعسفي تماما . وسيكون من الممكن تماما اختيار اتفاقيات أخرى مختلفة تماما . يمكنك بالطبع انتقاد الاختيارات، وربما ترغب في استبدالها بأخرى أكثر فاعلية أو أبسط .

أنت تعرف مسبقا أن الرسائل المتبادلة هي سلاسل بسيطة من وحدات البايتات . توقع أن هذه الرسائل سوف تنقل العديد من المعلومات في وقت واحد، ولقد قررنا أن كل واحد منهم يمكن أن يحمل العديد من المجالات، التي ستفصل بفواصل . عند استلام أي من هذه الرسائل، يمكننا بعد ذلك بسهولة استعادة جميع المكونات في القائمة، وذلك باستخدام أسلوب المدمج **split**().

هذا مثال عن نوع تحاور، كما يمكن اتباعها على جانب العميل . الرسائل النجمية هي التي وردة من الخادم، والأخرى هي تلك الصادرة من قبل العميل نفسه :

```
 1#  *serveur OK*
 2#  client OK
 3#  *canons,Thread-3;104;228;1;dark red,Thread-2;454;166;-1;dark blue,*
 4#  OK
 5#  *nouveau_canon,Thread-4,481,245,-1,dark green,le_vôtre*
 6#  orienter,25,
 7#  feu
 8#  *mouvement_de,Thread-4,549,280,*
 9#  feu
10#  *mouvement_de,Thread-4,504,278,*
11#  *scores,Thread-4;1,Thread-3;-1,Thread-2;0,*
12#  *angle,Thread-2,23,*
13#  *angle,Thread-2,20,*
14#  *tir_de,Thread-2,*
15#  *mouvement_de,Thread-2,407,191,*
16#  *départ_de,Thread-2*
17#  *nouveau_canon,Thread-5,502,276,-1,dark green*
```

- عندما يبدأ عميل جديد،يرسل طلب اتصال إلى الخادم، الذي يرسل له رسالة عودة : "serveur OK" . عند استلام هذا الأخير، سيستجب العميل من خلال إرسال "client OK" . هذا الأول تبادل مجملات والتي هي ليست ضرورية، لكنه يضمن أن البلاغ على ما يرام في كلا الاتجاهين . ذلك يحذر أن العميل على استعداد للعمل، سيقوم الخادم بإرسال وصف للمدافع بالفعل في اللعبة (إن وجدت) : اسم المستخدم، الموقع على اللوحة. التوجه، واللون (السطر 3) .
- ردا على استلام العميل (السطر 4)، سيقوم الخادم بتثبيت مدفع جديد في فضاء اللعبة، فهو يشير إلى خصائص التثبيت، وليس فقط للعميل الذي تسبب بذلك، ولكن حتى جميع العملاء الآخرين المتصلين . والرسالة المرسلة للعميل الجديد تحمل فرق (لأنه هو صاحب مدفع الجديد) : بالإضاف إلى المميزات الموجودة في المدفع، والتي يتم توفيرها للجميع، ولديه حقل إضافي يحتوي ببساطة على "le_vôtre" (قارن على سبيل المثال بين السطر 5 مع السطر 17، مما يدل على اتصاله من لاعب أخر) . هذه الإشارة الإضافية تسمح للعميل بتمييز بين مدفع، في رسائل عدة متماثلة، والتي تحتوي على تعريف موحد مسند إلى الخادم .

- الرسائل في السطور 6 و 7 هي أوامر مرسلة من قبل العميل (إنشاء والتحكم في إطلاق النار) . في النسخة السابقة من اللعبة، كنا قد وافقنا على أن المدافع ستتحرك قليلا (و بشكل عشوائي) بعد كل طلقة . بالتالي فإن الخادم هو الذي سيقوم بهذه العملية، ثم يقوم بإرسال النتائج إلى كافة المستخدمين المتصلين . الرسالة الواردة إلى الخادم في السطر 8 هي مؤشر على هذه الخطوة (الإحداثيات المقدمة هي إحداثيات ناتجة عن المدفع المعني) .
- السطر 11 يقوم بنسخ نوع الرسالة المرسلة من قبل الخادم عندما يتم ضرب الهدف . وترسل أيضا نتائج اللاعبين الجديدة إلى جميع اللاعبين لجميع العملاء .
- رسائل الخادم في الأسطر 12 و 13 و 14 تشير إلى الإجراءات التي اتخذها اللاعب الأخر (إعداد تتبع إطلاق النار) . مرة أخرى، يتم نقل المدفع عشوائيا بعد كل إطلاق نار (السطر 15) .
- السطران 16 و 17 : عندما يقطع أحد العملاء الاتصال، يقوم الخادم بإعلام العملاء الآخرين، بإختفاء المدفع في فضاء اللعب على جميع الأماكن . وعلى العكس، يمكن للعملاء الجدد أن يقوموا باتصال جديد في أي وقت للعب اللعبة .

ملاحظات إضافية

الحقل الأول من كل رسالة يشير إلى مضمونها . الرسائل المرسلة من العميل بسيطة جدا : فهي تتوافق مع الإجراءات التي اتخذها معظم اللاعبين (تغيير في زاوية الإطلاق والسيطرة على النار) . والتي تم إرسالها من قبل الخادم هي قليلا أكثر تعقيدا . يتم إرسال معظمهم إلى كافة المستخدمين المتصلين للحفاظ على التقدم المحرز . ونتيجة ذلك، يجب أن تكون لهذه الرسائل معرف اللاعب الذي يسيطر على العمل أو يتأثر أو يغير. لقد رأينا أعلاه أن هذه المعرفات هي أسماء تم صنعها تلقائيا من قبل خادم المواضيع، في كل مرة يتصل بها عميل جديد .

بعض رسائل اللعبة تحتوي على معلومات بالمجال . في هذه الحالة، التغييرات "المجالات-الفرعية" تكون مفصولة بفواصل منقوطة (السطران 3 و 11) .

برنامج خادم : الجزء الأول

سوف نجد في الصفحات التالية سكريبت كامل لبرنامج خادم . سوف نقدم إليك في ثلاثة قطع على التوالي تقليقات الكود، ولكن ترقيم الأسطر مستمر . على الرغم من أنه بالفعل طويل جدا ومعقد، سوف تشعر أنك تستحق ربما المزيد من التحسين، لا سيما من حيث العرض العام . نترك لك الأمر للإضافة بنفسك جميع الملاحق التي قد تبدو مفيدة (على سبيل المثال، اقتراح لاختيار إحداثيات الجهاز المصيف عند بدء التشغيل، وشريط قوائم وإلخ) :

```
#######################################################
# لعبة القصف - نسخة الخادم                            #
# (C) Gérard Swinnen, Verviers (Belgique) - July 2004 #
# رخصة  : GPL                              rév. 2010 #
# قبل تشغيل هذا السكريبت، تأكد من أن عنوان      #
```

```
# IP في الأسفل نفس في جهازك.                        #
# يمكنك اختيار رقم منفذ اخر، أو              #
# تغيير إحداثيات فضاء اللعبة.                       #
# في جميع الأحوال، تأكد من أن نفس الاختيارات      #
# تم وضعها في كل سكريبت عميل.                          #
#######################################################

host, port = '192.168.1.168', 36000
largeur, hauteur = 700, 400            # إحداثيات فضاء اللعبة

from tkinter import *
import socket, sys, threading, time
import canon03
from canon04 import Canon, AppBombardes

class Pupitre(canon03.Pupitre):
    """Pupitre de pointage amélioré"""
    def __init__(self, boss, canon):
        canon03.Pupitre.__init__(self, boss, canon)

    def tirer(self):
        "déclencher le tir du canon associé"
        self.appli.tir_canon(self.canon.id)

    def orienter(self, angle):
        "ajuster la hausse du canon associé"
        self.appli.orienter_canon(self.canon.id, angle)

    def valeur_score(self, sc =None):
        "imposer un nouveau score <sc>, ou lire le score existant"
        if sc == None:
            return self.score
        else:
            self.score =sc
            self.points.config(text = ' %s ' % self.score)

    def inactiver(self):
        "désactiver le bouton de tir et le système de réglage d'angle"
        self.bTir.config(state =DISABLED)
        self.regl.config(state =DISABLED)

    def activer(self):
        "activer le bouton de tir et le système de réglage d'angle"
        self.bTir.config(state =NORMAL)
        self.regl.config(state =NORMAL)

    def reglage(self, angle):
        "changer la position du curseur de réglage"
        self.regl.config(state =NORMAL)
        self.regl.set(angle)
        self.regl.config(state =DISABLED)
```

• الصنف **()Pupitre** تم بناؤه عن طريق الاشتقاق من صنف لديه نفس الاسم تم استدعاؤه من وحدة **canon03**. ولذلك فإنه[111] يرث جميع خصائصه، لكن نحن بحاجة لتجاوز أساليبه **()tirer** و **()orienter**.

[111] تذكير : في الصنف المشتق، يمكنك تعريف أسلوب جديد مع نفس اسم أسلوب الصنف الأصل، لتغيير وظائفها في الصنف المشتق . و هذا يسمى تجاوز هذا الأسلوب (انظر أيضا إلى صفحة 196).

- في الإصدار الفردي للبرنامج، في الواقع، يمكنك التحكم بكل مدفع من خلال لوحة التحكم . في نسخة الشبكة، العميل هو الذي سيقوم بالتحكم عن بعد بتشغيل المدافع . ولذلك، العميل هو الذي يتحكم عن بعد بالمدفع . ولا يمكن للواحات التحكم التي تظهر في نافذة الخادم بتكرار المناورات التي يقوم بها اللاعبون من خلال كل عميل . زر الإطلاق ومؤشر الإرتفاع معطلان (غير مفعلان)، لكن المؤشرات تطيع الأوامر التي تم إرسالها من قبل التطبيق الرئيسي .
- هذا الصنف الجديد **Pupitre()** سوف يستخدم في كل نسخة من برنامج العميل . في نافذته مثل التي في الخادم، جميع لوحات التحكم تظهر كمكررات، لكن واحد منهم سيكون جاهزا تماما : الذي يتوافق مع مدفع اللاعب .
- كل هذه الأسباب تؤدي إلى ظهور أساليب جديدة **reglage**, **desactiver()**, **activer()** و **valeur_score()**, الذين سيتم استدعاؤهم من قبل التطبيق الرئيسي، ردا على الرسائل المتبادلة بين الخادم وعملائه .
- يتم استخدام الصنف **ThreadConnexion()** أدناه لإنشاء مثيل لمجموعة من الكائنات المواضيع التي تتسلم بالتوازي جميع الاتصالات من العملاء . يحتوي أسلوبه **run()** على الوظائف الأساسية للخادم، حلقة التعليمات التي تدير استقبال الرسائل من عميل معين، والتي لكل واحدة منها سلسلة من ردود الفعل . سوف تجد تنفيذ ملموس لبروتوكول الاتصال الموضح في الصفحات السابقة (بعض الرسائل تم صنعها بواسطة الأساليب **depl_aleat_canon()** و **goal()** من صنف **AppServeur()** الذي سيتم شرحه لاحقا).

```
class ThreadConnexion(threading.Thread):
    """objet thread gestionnaire d'une connexion client"""
    def __init__(self, boss, conn):
        threading.Thread.__init__(self)
        self.connexion = conn            # مرجع socket الاتصال
        self.app = boss                  # مرجع نافذة التطبيق

    def run(self):
        "actions entreprises en réponse aux messages reçus du client"
        nom = self.getName()             # معرف العميل = اسم الخيط
        while 1:
            msgClient = self.connexion.recv(1024).decode("Utf8")
            print("**{0}** de {1}".format(msgClient, nom))
            deb = msgClient.split(',')[0]
            if deb == "fin" or deb =="":
                self.app.enlever_canon(nom)
                # علامة بداية هذا المدفع للعملاء الآخرين :
                self.app.verrou.acquire()
                for cli in self.app.conn_client:
                    if cli != nom:
                        message = "départ_de,{0}".format(nom)
                        self.app.conn_client[cli].send(message.encode("Utf8"))
                self.app.verrou.release()
                # إغلاق هذا الموضوع :
                break
            elif deb =="client OK":
                # علامة إلى عميل جديد أن المداقع مسجلة بالفعل :
```

```
85#                 msg ="canons,"
86#                 for g in self.app.guns:
87#                     gun = self.app.guns[g]
88#                     msg =msg +"{0};{1};{2};{3};{4},".\
89#                         format(gun.id, gun.x1, gun.y1, gun.sens, gun.coul)
90#                 self.app.verrou.acquire()
91#                 self.connexion.send(msg.encode("Utf8"))
92#                 # انتظار الحصول على اعتراف ('OK') :
93#                 self.connexion.recv(100).decode("Utf8")
94#                 self.app.verrou.release()
95#                 # إضافة مدفع إلى فضاء لعب الخادم.
96#                 # الأسلوب الذي تم استدعاؤه يرجع صفات للمدفع الذي صُنع :
97#                 x, y, sens, coul = self.app.ajouter_canon(nom)
98#                 # إرسال الصفات لهذا المدفع الجديد إلى كل العملاء المتصلة
99#
100#                self.app.verrou.acquire()
101#                for cli in self.app.conn_client:
102#                    msg ="nouveau_canon,{0},{1},{2},{3},{4}".\
103#                        format(nom, x, y, sens, coul)
104#                    # للعميل الجديد، أضف حقل مشير إلى أن
105#                    # الرسالة تتعلق بالمدفع الخاص بها :
106#                    if cli == nom:
107#                        msg =msg +",le_vôtre"
108#                    self.app.conn_client[cli].send(msg.encode("Utf8"))
109#                self.app.verrou.release()
110#            elif deb =='feu':
111#                self.app.tir_canon(nom)
112#                # الإشارة إلى هذا الإطلاق للنار لجميع العملاء الاخرين :
113#                self.app.verrou.acquire()
114#                for cli in self.app.conn_client:
115#                    if cli != nom:
116#                        message = "tir_de,{0},".format(nom)
117#                        self.app.conn_client[cli].send(message.encode("Utf8"))
118#                self.app.verrou.release()
119#            elif deb =="orienter":
120#                t =msgClient.split(',')
121#                # يمكن أن نستقبل العديد من الزوايا. نقوم باستخدام الأخيرة :
122#                self.app.orienter_canon(nom, t[-1])
123#                # الإشارة لهذه التغييرات لجميع العملاء الاخرين :
124#                self.app.verrou.acquire()
125#                for cli in self.app.conn_client:
126#                    if cli != nom:
127#                        # نقطة نهاية لأن، الرسائل قد تكون مجمعة في بعض الأحيان :
128#                        message = "angle,{0},{1},".format(nom, t[-1])
129#                        self.app.conn_client[cli].send(message.encode("Utf8"))
130#                self.app.verrou.release()
131#
132#        # إغلاق الاتصال :
133#        self.connexion.close()            # قطع الاتصال
134#        del self.app.conn_client[nom]     # حذف مرجعه في القاموس
135#        self.app.afficher("Client %s déconnecté.\n" % nom)
136#        # الخيط ينتهي هنا
```

تزامن الخيوط باستخدام الأقفال (thread locks)

من خلال فحص الكود أعلاه، سوف تلاحظ بنية معينة من كتل التعليمات التي يرسل الخادم نفس لرسالة إلى جميع عملائه . على سبيل المثال انظر إلى الأسطر من 74 إلى 80 .

السطر 75 يقوم بنشيط الأسلوب **acquire()** لكائن "مغلق" تم إنشاؤه عن طريق المنشئ للتطبيق الرئيسي (انظر أدناه) . هذا الكائن هو مثيل الصنف **Lock()**, الذي هو جزء من وحدة **threading** التي قمنا باستدعائها في بداية السكريبت . الأسطر التالية (من 76 إلى 79) تسبب إرسال رسالة إلى كافة العملاء المتصلين (باستثناء واحد) . ثم، يكون لكائن-القفل مسعى جديد، هذه المرة لأسلوبه **release()**.

ماذا يخدم هذا كائن-القفل إذا ؟ بما أنه يتم صنعه من صنف من وحدة **threading**, يمكنك تخمين أن له فائدة للمواضيع . في الواقع، تستخدم هذه كائنات-القفل لمزامنة المواضيع المتزامنة . ما هو ؟هل تعلم أن الخادم يبدأ بموضوع مختلف لكل عميل متصل . ثم، تصبح كل هذه المواضيع تعمل بالتوازي . وبالتالي هنالك خطر في بعض الأحيان، وهو ربما أن موضوعين أو أكثر يمكن أن يستخدموا موردا مشتركا في نفس الوقت .

في الأسطر البرمجية التي ناقشناها، على سبيل المثال، نحن نتعامل مع موضوع يريد تقريبا استخدام جميع الاتصالات الموجودة لنشر الرسالة . ولذلك من الممكن أنه خلال هذا الوقت، موضوع آخر يريد أن يستخدم أيضا واحدة أو أخرى من هذه الاتصالات، والتي قد يتسبب في عطب (أي تطابق بشكل فوضوي عدة رسائل) .

يمكن حل هذه المشكلة باستخدام كائن-القفل (thread lock) . هذا الكائن لا يتم إنشاؤه إلا في نسخة واحدة، في مساحة الأسماء التي في متناول جميع المواضيع المتزامنة . ويتميز أساسا في أنه دائما في حالة أو أخرى : مغلق أو غير مغلق . حالته الأولية هي غير مغلق .

الاستخدام

عندما يكون أي موضوع على وشك الوصول إلى مورد مشترك، يتم أولا تفعيل الأسلوب **acquire()** للقفل . فإذا كانت الحالة غير مغلق، يتم غلقه، ويمكن للموضوع الذي طلبه أن يستخدم مصادر مشتركة بأمان . عند ااإنتهاء من استخدام المورد، فإنه سوف يقوم بتفعيل الأسلوب **release()** للقفل، الذي سيقوم بفتح قفله (يكون غير مغلق) .

في الواقع، إذا كان موضوع أخر يحاول تفعيل هو أيضا الأسلوب **acquire()** للقفل، عندما يكون في حالة مقفل، يكون الأسلوب "ليس في متناول يده"، مما يتسبب في منع هذا الموضوع، والذي يقوم بتعليق نشاطه حتى يعود إلى حالت غير مقفل . وهذا بالتالي يمنع الوصول إلى مورد مشترك خلال الوقت الذي يستخدم فيه موضوع آخر . عندما يتم فتح القفل، واحدة من المواضيع الانتظار (قد يكون في الواقع أكثر من واحد) يستأنف نشاطه أثناء إغلاق القفل، وهكذا .

يقوم كائن-القفل بحفظ مراجع المواضيع الذي قد منعها، بحيث أنه يتم إزالة المنع لواحد فقط عندما يتم استدعاء الأسلوب **release()** ويجب دائما لكل موضوع يتم تفعيله باستخدام الأسلوب **acquire()** قبل الوصول إلى الموارد، ويجب عليه نفعيل أسلوبه **release()** بعد ذلك .

شرط أن تكون جميع المواضيع المتزامنة تتبع نفس الإجراء، هذه التقنية بسيطة تمنع إمكانية استخدام مورد مشترك في وقت واحد من قبل العديد منهم . نقول قي هذه الحالة المواضيع قد تم مزامنتها .

برنامج الخادم : إنهائه

الصنفان بالأسفل تكمل السكريبت الخادم . يتم تنفيذ التعليمات البرمجية في الصنف **ThreadClients()** مماثلة لتلك التي طورناها سابقا لجسم تطبيق برنامج الدردشة . في هذه الحالة، ومع ذلك، وضعنا في صنف مشتق من **Thread()**، لأنه يحتاج إلى تشغيل هذا الكود في موضوع مستقل عن التطبيق الرئيسي . وبالفعل تم القبض عليه عن طريق حلقة **mainloop()** للواجهة الرسومية[112].

يشتق الصنف **AppServeur()** من صنف **AppBombardes()** لوحدة **canon04**. ولقد أضفنا مجموعة من الأساليب المكملة للقيام بتنفيذ جميع العمليات التي تنتج عن حوار مع العملاء . لاحظنا سابقا أن العملاء يمثلون نسخة مشتقة من هذا الصنف (تمتع بنفس التعاريف الأساسية للنافذة، اللوحة، إلخ) .

```
class ThreadClients(threading.Thread):
    """objet thread gérant la connexion de nouveaux clients"""
    def __init__(self, boss, connex):
        threading.Thread.__init__(self)
        self.boss = boss                # مرجع نافذة التطبيق
        self.connex = connex            # مرجع socket المبدئي

    def run(self):
        "attente et prise en charge de nouvelles connexions clientes"
        txt ="Serveur prêt, en attente de requêtes ...\n"
        self.boss.afficher(txt)
        self.connex.listen(5)
        # إدارة الاتصالات المطلوبة من قبل العملاء :
        while 1:
            nouv_conn, adresse = self.connex.accept()
            # صنع كائن خيط جديد لصنع الاتصال :
            th = ThreadConnexion(self.boss, nouv_conn)
            th.start()
            it = th.getName()         # معرف فريد للخيط
            # حفظ الاتصال في قاموس :
            self.boss.enregistrer_connexion(nouv_conn, it)
            # إظهاره :
            txt = "Client %s connecté, adresse IP %s, port %s.\n" %\
                   (it, adresse[0], adresse[1])
            self.boss.afficher(txt)
            # بدء التواصل مع العميل :
            nouv_conn.send("serveur OK".encode("Utf8"))

class AppServeur(AppBombardes):
    """fenêtre principale de l'application (serveur ou client)"""
    def __init__(self, host, port, larg_c, haut_c):
        self.host, self.port = host, port
```

112 سوف نناقش هذا السؤال في الصفحات القادمة، لأنه يفتح بعض الوجهات المثيرة للاهتمام : انظر إلى : تحسين الرسوم المتحركة باستخدام الخيوط، صفحة 405.

```
        AppBombardes.__init__(self, larg_c, haut_c)
        self.active =1          # مؤشر النشاط
        # تأكد من خروجك بشكل صحيح عند إغلاق النافذة :
        self.bind('<Destroy>',self.fermer_threads)

    def specificites(self):
        "préparer les objets spécifiques de la partie serveur"
        self.master.title('<<< Serveur pour le jeu des bombardes >>>')

        # ويدجت نص، مرتبط مع شريط تمرير :
        st =Frame(self)
        self.avis =Text(st, width =65, height =5)
        self.avis.pack(side =LEFT)
        scroll =Scrollbar(st, command =self.avis.yview)
        self.avis.configure(yscrollcommand =scroll.set)
        scroll.pack(side =RIGHT, fill =Y)
        st.pack()

        # جزء خادم الاتصال :
        self.conn_client = {}           # قاموس اتصالات العميل
        self.verrou =threading.Lock()   # قفل مزامنة الخيوط
        # وضع - تهيئة الخادم socket :
        connexion = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        try:
            connexion.bind((self.host, self.port))
        except socket.error:
            txt ="La liaison du socket à l'hôte %s, port %s a échoué.\n" %\
                  (self.host, self.port)
            self.avis.insert(END, txt)
            self.accueil =None
        else:
            # بدء خيط لمراقبة اتصال العملاء :
            self.accueil = ThreadClients(self, connexion)
            self.accueil.start()

    def depl_aleat_canon(self, id):
        "déplacer aléatoirement le canon <id>"
        x, y = AppBombardes.depl_aleat_canon(self, id)
        # الإشارة لهذه الإحداثيات الجديدة لكافة العملاء :
        self.verrou.acquire()
        for cli in self.conn_client:
            message = "mouvement_de,%s,%s,%s," % (id, x, y)
            self.conn_client[cli].send(message.encode("Utf8"))
        self.verrou.release()

    def goal(self, i, j):
        "le canon <i> signale qu'il a atteint l'adversaire <j>"
        AppBombardes.goal(self, i, j)
        # الإشارة إلى النتائج الجديدة لكافة العملاء :
        self.verrou.acquire()
        for cli in self.conn_client:
            msg ='scores,'
            for id in self.pupi:
                sc = self.pupi[id].valeur_score()
                msg = msg +"%s;%s," % (id, sc)
            self.conn_client[cli].send(msg.encode("Utf8"))
        time.sleep(.5)                  # للتحسين نفصل الرسائل
        self.verrou.release()

    def ajouter_canon(self, id):
        "instancier un canon et un pupitre de nom <id> dans 2 dictionnaires"
```

```
        # on alternera ceux des 2 camps :
        n = len(self.guns)
        if n %2 ==0:
            sens = -1
        else:
            sens = 1
        x, y = self.coord_aleat(sens)
        coul =('dark blue', 'dark red', 'dark green', 'purple',
               'dark cyan', 'red', 'cyan', 'orange', 'blue', 'violet')[n]
        self.guns[id] = Canon(self.jeu, id, x, y, sens, coul)
        self.pupi[id] = Pupitre(self, self.guns[id])
        self.pupi[id].inactiver()
        return (x, y, sens, coul)

    def enlever_canon(self, id):
        "retirer le canon et le pupitre dont l'identifiant est <id>"
        if self.active == 0:        # تم إغلاق النافذة
            return
        self.guns[id].effacer()
        del self.guns[id]
        self.pupi[id].destroy()
        del self.pupi[id]

    def orienter_canon(self, id, angle):
        "régler la hausse du canon <id> à la valeur <angle>"
        self.guns[id].orienter(angle)
        self.pupi[id].reglage(angle)

    def tir_canon(self, id):
        "déclencher le tir du canon <id>"
        self.guns[id].feu()

    def enregistrer_connexion(self, conn, it):
        "Mémoriser la connexion dans un dictionnaire"
        self.conn_client[it] = conn

    def afficher(self, txt):
        "afficher un message dans la zone de texte"
        self.avis.insert(END, txt)

    def fermer_threads(self, evt):
        "couper les connexions existantes et fermer les threads"
        # قطع الاتصالات مع جميع العملاء :
        for id in self.conn_client:
            self.conn_client[id].send('fin'.encode("Utf8"))
        # فرض إنهاء خيط خادم انتظار الإستعلامات (الطلبات) :
        if self.accueil != None:
            self.accueil._stop()
        self.active =0                  # منع وصول إلى لاحقة Tk

if __name__ =='__main__':
    AppServeur(host, port, largeur, hauteur).mainloop()
```

تعليقات

- السطر 173 : سيحدث من وقت لأخر يريد "الاعتراض " على إغلاق التطبيق الذي قام المستخدم بالخروج من برنامجك، على سبيل المثال لأنك ترير إجباره بحفظ نسبة من البيانات المهمة في ملف، أو غلق نوافذ أخرى، إلخ . فقط اجعله يكشف عن العنصر <Destroy>، كما فعلنا هنا لإجبار إنهاء جميع المواضيع النشطة .
- *السطور من 179 إلى 186 : هنا عليك مراجعة تقنية ربط شريط التمرير مع ويدجت "نص - **Text**" (انظر للصفحة247).
- السطر 190 : إنشاء مثيل لكائن-قفل يسمح بمزامنة المواضيع .
- السطران 202 و 203 : هنا يتم تمثيل كائن الموضوع الذي ينتظر بشكل مستمر طلبات اتصال العملاء المحتملين .
- السطور من 205 إلى 213 ومن 215 إلى 227 : هذه الأساليب تجاوز الأساليب من نفس الاسم التي ورثت من صنف الأصل . فهي تبدأ باستدعائهم للقيام بنفس العمل (السطور من 207 إلى 217)، ومن ثم إضافة وظائفهم الخاصة، وهو الذي يقدم الجميع ما حدث الآن .
- السطور من 229 إلى 243 : هذا الأسلوب يمثل موقف جديد للإطلاق في كل مرة يتصل فيها عميل جديد . يتم وضع المدافع بالتناوب في الجانب الأيمن والأيسر، وهو إجراء من الواضح أنه يمكن أن يتحسن . قائمة الألوان الموجود تحد عدد العملاء إلى 10، والذي ينبغي أن يكون كافيا .

برنامج العميل

سكريبت برنامج العميل أدناه . كما في الخادم، فهو قصير نسبيا، لأنه يستخدم أيضا استدعاء الوحدات ويرث الأصناف . يجب على سكريبت الخادم أن يتم حفظه في ملف-وحدة تسمى **canon_serveur.py**. وهذا الملف يجب وضعه في الدليل الحالي، وكذلك ملفات-الوحدات **canon03.py** و **canon04.py** الذي يستخدمهم .

هذه الوحدات تم استدعاؤها، هذا السكريبت يستخدم الأصناف **Canon**() و **Pupitre**() بشكل مطابق، والمشتقة من صنف **AppServeur**(). في هذه الأخيرة، العديد من الأساليب الثقيلة للتكيف مع وظائفها . على سبيل المثال الأساليب **goal**() و **depl_aleat_canon**()، لا تفعل شيئا (العبارة **pass**)، لأنه لا يمكن حساب النتائج وإعادة تموضع المدافع بعد كل إطلاق للنار لأن هذا يتم من الخادم فقط .

سوف تجد في الأسلوب **run**() للصنف **ThreadSocket**() (الأسطر من 86 إلى 126) كود الذي يعالج الرسائل المتبادلة مع الخادم . ونحن أبقينا تعليمة **print** (في السطر 88) ذلك لأن الرسائل المتلقاة من الخادم تظهر على الإخراج القياسي . فإذا قمت بنفسك بعمل شكل أكثر تحديدا لهذه اللعبة، يمكنك بالطبع حذف هذه التعليمات .

```
#####################################################
# لعبة القصف – نسخة العميل                         #
# (C) Gérard Swinnen, Liège (Belgique) - Juillet 2004 #
# الرخصة : GPL                          Révis.  2010 #
# قبل تشغيل هذا السكريبت، تأكد من عنوان              #
# رقم المنفذ وإحداثيات فضاء                          #
# اللعبة التي تم وضعها في الأسفل هي نفسها بالضبط #
# التي تم تعريفها للخادم.                             #
#####################################################

from tkinter import *
import socket, sys, threading, time
from canon_serveur import Canon, Pupitre, AppServeur

host, port = '192.168.1.168', 36000
largeur, hauteur = 700, 400          # إحداثيات فضاء اللعب

class AppClient(AppServeur):
    def __init__(self, host, port, larg_c, haut_c):
        AppServeur.__init__(self, host, port, larg_c, haut_c)

    def specificites(self):
        "préparer les objets spécifiques de la partie client"
        self.master.title('<<< Jeu des bombardes >>>')
        self.connex =ThreadSocket(self, self.host, self.port)
        self.connex.start()
        self.id =None

    def ajouter_canon(self, id, x, y, sens, coul):
        "instancier un canon et un pupitre de nom <id> dans 2 dictionnaires"
        self.guns[id] = Canon(self.jeu, id, int(x), int(y), int(sens), coul)
        self.pupi[id] = Pupitre(self, self.guns[id])
        self.pupi[id].inactiver()

    def activer_pupitre_personnel(self, id):
        self.id =id                           # تلقي معرف من الخادم
        self.pupi[id].activer()

    def tir_canon(self, id):
        r = self.guns[id].feu()               # إذا توقف False إرسال
        if r and id == self.id:
            self.connex.signaler_tir()

    def imposer_score(self, id, sc):
        self.pupi[id].valeur_score(int(sc))

    def deplacer_canon(self, id, x, y):
        "note: les valeurs de x et y sont reçues en tant que chaînes"
        self.guns[id].deplacer(int(x), int(y))

    def orienter_canon(self, id, angle):
        "régler la hausse du canon <id> à la valeur <angle>"
        self.guns[id].orienter(angle)
        if id == self.id:
            self.connex.signaler_angle(angle)
        else:
            self.pupi[id].reglage(angle)

    def fermer_threads(self, evt):
        "couper les connexions existantes et refermer les threads"
        self.connex.terminer()
```

```
        self.active =0                     # منع وصول إلى لاحقة Tk

    def depl_aleat_canon(self, id):
        pass                               # => أسلوب غير فعال

    def goal(self, a, b):
        pass                               # => أسلوب غير فعال


class ThreadSocket(threading.Thread):
    """objet thread gérant l'échange de messages avec le serveur"""
    def __init__(self, boss, host, port):
        threading.Thread.__init__(self)
        self.app = boss              # مرجع نافذة التطبيق
        # وضع socket - : اتصال بالخادم :
        self.connexion = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        try:
            self.connexion.connect((host, port))
        except socket.error:
            print("La connexion a échoué.")
            sys.exit()
        print("Connexion établie avec le serveur.")

    def run(self):
        while 1:
            msg_recu = self.connexion.recv(1024).decode("Utf8")
            print("*%s*" % msg_recu)
            # le message reçu est d'abord converti en une liste :
            t =msg_recu.split(',')
            if t[0] =="" or t[0] =="fin":
                # حذف هذا الخيط :
                break
            elif t[0] =="serveur OK":
                self.connexion.send("client OK".encode("Utf8"))
            elif t[0] =="canons":
                self.connexion.send("OK".encode("Utf8"))   # اعتراف وصول
                # مسح أول واخر عنصر في القائمة.
                # تبقي قوائم :
                lc = t[1:-1]
                # كل واحد هو وصف كامل للمدفع :
                for g in lc:
                    s = g.split(';')
                    self.app.ajouter_canon(s[0], s[1], s[2], s[3], s[4])
            elif t[0] =="nouveau_canon":
                self.app.ajouter_canon(t[1], t[2], t[3], t[4], t[5])
                if len(t) >6:
                    self.app.activer_pupitre_personnel(t[1])
            elif t[0] =='angle':
                # من الممكن أن نستقبل مجموعة من المعلومات المجمعة.
                # لن نأخذ سوى الأولى :
                self.app.orienter_canon(t[1], t[2])
            elif t[0] =="tir_de":
                self.app.tir_canon(t[1])
            elif t[0] =="scores":
                # مسح أول واخر عنصر في القائمة.
                # تبقي قوائم :
                lc = t[1:-1]
                # كل عنصر هو وصف نتيجة :
                for g in lc:
                    s = g.split(';')
                    self.app.imposer_score(s[0], s[1])
```

```
123#                elif t[0] =="mouvement_de":
124#                    self.app.deplacer_canon(t[1],t[2],t[3])
125#                elif t[0] =="départ_de":
126#                    self.app.enlever_canon(t[1])
127#
128#            # خيط <réception> ينتهي هنا.
129#            print("Client arrêté. Connexion interrompue.")
130#            self.connexion.close()
131#
132#        def signaler_tir(self):
133#            self.connexion.send("feu".encode("Utf8"))
134#
135#        def signaler_angle(self, angle):
136#            msg ="orienter,{0}".format(angle)
137#            self.connexion.send(msg.encode("Utf8"))
138#
139#        def terminer(self):
140#            self.connexion.send("fin".encode("Utf8"))
141#
142#    # Programme principal :
143#    if __name__ =='__main__':
144#        AppClient(host, port, largeur, hauteur).mainloop()
```

تعليقات

- السطران 15 و 16 : يمكنك تحسين هذا السكريبت بإضافة نموذج لطلب هذه القيم من المستخدم أثناء بدء التشغيل .
- * السطور من 19 إلى 27 : ينتهي منشئ صنف الأصل باستدعاء الأسلوب **specificites()**. يمكننا أن نضع فيها ما يجب أن يبني بشكل مختلف في الخادم والعملاء . يقوم الخادم خاصة بتمثيل ويدجت "نص - **text** " الذي لم تظهر في العملاء، واحدة أو أخرى من الكائنات الموضوع المختلفة تبدأ لإدارة الاتصالات .
- * السطور من 39 إلى 42 : يتم استدعاء هذا الأسلوب في كل مرة يقوم فيها المستخدم بالضغط على زر إطلاق النار . لا يمكن للمدفع إلا لقطات بشكل مستمر . لذلك، لا يمكن لأي رصاصة جديدة أن تقبل إذا لم تكمل الطلقة التي قبلها مسارها، القيم "صحيح" أو "خطأ" تقوم بإرجاع باستخدام الأسلوب **feu()** لكائن المدفع الذي يشير إلى ما إذا تم قبول إطلاق النار أو لا . يتم استخدام هذه القيمة للإشارة إلى الخادم (و عملاء آخرين) أن إطلاق النار قد حصل فعلا .
- * السطور من 105 إلى 108 : يتم إضافة مدفع جديد في فضاء لعبة لكل واحد (و هذا معناه في لوحة الخادم وفي لوحات جميع العملاء المتصلين)، في كل مرة يتصل فيها عميل جديد . يرسل الخادم في هذا الوقت رسالة إلى جميع العملاء لإعلامهم بهذا الشريك الجديد . لكن الرسالة ترسل على وجه الخصوص تحمل حقلا إضافيا (يحتوي ببساطة على سلسلة "le_vôtre") بحيث أن هذا الشريك يعلم أن هذه الرسالة لمدفع خاص به، ويمكن تفعيل لوحة التحكم أثناء تخزين المعرف الذي تم تعيينه من قبل الخادم (انظر أيضا إلى السطور من 35 إلى 37) .

استنتاجات ووجهات نظر

تم تقديم هذا التطبيق في هدف تعليمي . لقد تعمدنا تبسيط العديد من المشاكل . على سبيل المثال، إذا إختبرت بنفسك هذا البرنامج، سوف تجد أنه في كثير من الأحيان تم تجميع الرسائل المتبادلة في "الحزم"، وهذا يتطلب تحسين خوارزمية تنفيذ لتفسيرها .

و بالمثل، فإننا بالكاد رسمنا الآلية الأساسية للعبة : توزيع اللاعبين على معسكرين، وتدمير المدافع المعنية، ومختلف العقبات الأخرى، إلخ . ولدينا العديد من الطرق لاستكشافها !

تمارين

1.19 قم بتبسيط السكريبت لعميل الدردشة الذي تم شرحه في الصفحة 384, عن طريق إزالة كائن من كائني المواضيع . وقم على سبيل المثال بترتيب معالجة إرسال الرسائل في الموضوع الرئيسي .

2.19 قم بتعديل لعبة القصف (الإصدار المستقل) في الفصل 15 (انظر إلى صفحات 276 وما يليها)، بحفظ لوحة تحكم واحدة ومدفع واحد . وأضف هدفا متحركا، وحركات سيتم إدراجها من كائن موضوع مستقل (من أجل فصل أجزاء من تعليمات البرمجية التي تحكم في حركة الهدف والمقذوف) .

استخدام المواضيع لتحسين الرسوم المتحركة

التمرين الأخير في نهاية المقطع السابق يشير إلى وجود منهجية لتطوير التطبيقات التي يمكن أن تكون مفيدة بشكل خاص في حالة ألعاب الفيديو التي تشمل العديد من الرسوم المتحركة المتزامنة .

في الواقع، إذا قمت ببرمجة مختلف عناصر الرسوم المتحركة للعبة ككائنات مستقاة تعمل كل واحدة منها على موضوع خاص بها، فأنت لا تبسط فقط هذه المهم بل حتى حسنت من إمكانية قراءة السكريبت الخاص بك، ولكن يمكنك زيادة سرعة التنفذ وبالتالي سيولة الرسوم المتحركة . لتحقيق هذه النتيجة، يجب عليك ترك مهلة للتقنية التي نستغلها حتى الآن، لكن سوف تستخدم في مكانها أسهل في نهاية المطاف .

تأخير الرسوم المتحركة باستخدام ()after

في جميع الرسوم المتحركة التي تحدثنا عنها حتى الآن، يتم صنع "المحرك" في كل مرة من خلال دالة تحتوي على أسلوب **()after**, تربط تلقائيا لجميع ويدجات tkinter، أنت تعلم أن هذا الأسلوب يسمح بإدخال تأخير وقت في سلوك برنامجك : يتم تنشيط مؤقت داخلي، بحيث بعد فترة من الوقت المتفق عليه، يستدعي النظام تلقائيا أي دالة . بشكل عام، الدالة التي تحتوي على **()after** يتم استدعاؤها : فهي توفر التالي في حلقة عودية، ويبقى برمجة مواقع للكائن الرسومية المختلفة .

اعلم أنه خلال تدفق الفاصل الزمني للبرنامج باستخدام الأسلوب **after()**, تطبيقك ليس "مجمدا" . يمكنك على سبيل المثال، خلال هذا الوقت، الضغط على زر، إعادة تحجيم النافذة، إدخال مدخلات من خلال لوحة المفاتيح، إلخ . لكن كيف يكون هذا ممكنا ؟88

و لقد سبق وأن ذكرنا مرات عديدة أن التطبيقات تشمل دائما محرك رسوم حديث "يعمل" بشكل مستمر في الخلفية : هذا الجهاز يبدأ عندما تشغل الأسلوب **mainloop()** لنافذتك الرئيسية . كما يشير اسمه بوضوح، هذا الأسلوب يطبق دائما حلقة تكرار، نفس نوع حلقة **while** التي تعرفها جيدا . يتم تضمين العديد من آليات في هذا "المحرك" . واحدة من هذه آليات هي الحصول على جميع الأحداث التي تحدث، وبعد صنع تقرير لهم باستخدام الرسائل الملائمة للبرامج التي تطلب ذلك (انظر إلى : برامج تتحكم بواسطة الأحداث، صفحة88), على أن تتخذ إجراءات تحكم أخرى في العرض، إلخ . عندما تقوم باستدعاء أسلوب **after()** لويدجت، سوف تستخدم في الواقع آلية توقيت تتكامل مع **mainloop()**, أيضا، ويقوم هذا المدير المركزي باستدعاء الدالة التي تريدها، بعد فترة زمنية معينة .

تقنية تحريك الرسوم المتحركة تستخدم الأسلوب **after()** هي الوحيدة الممكن تطبيق عملها على موضوع واحد، لأن الحلقة **mainloop()** توجه السلوك العام مثل هذا الطلب إلى ذلك . هذا خاصة سيتولى إعادة رسم جزء أو كل النافذة كلما كان ذلك ضروريا . لهذا السبب لا يمكنك تخيل بناء محرك رسوم متحركة من شأنها أن تعيد تحديد إحداثيات الكائن الرسومي داخل حلقة **while**, بسيطة، على سبيل المثال، لأن في أثناء تشغيل **mainloop()** سوف تبقى معلقة، الأمر الذي يعني أنه خلال هذه الفترة الزمنية سوف يتم إعادة تصميم أي كائن رسومي (خاصة إذا كنت ترغب في الحصول على هذه الخطوة) . في الواقع، سوف يتم تجميد التطبيق، طالما لا يتم مقاطعة حلقة **while** .

هو الوحيد الممكن، هذه التقنية التي استخدمناها حتى الآن في أمثلة تطبيق الموضوع-الواحد الخاص بنا . لديها عيب واحد مزعج : نظرا لوجود عدد كبير من عمليات الدعم في كل تكرار لحلقة **mainloop()**, جهاز ضبط الوقت الذي يمكن برمجته باستخدام **after()** لا يمكن أن يكون قصيرا جدا . على سبيل المثال، لا يمكنها أن تقع أقل من 15 مللي ثانية بالكاد على حاسوب نموذجي (سنة 2004، بروسيسور من نوع Pentium IV، f = 1،5 GHz) . يجب أن تنظر إلى هذا القيد إذا كنت ترغب في تطوير رسوم متحركة سريعة .

عيب أخر عند استخدام الأسلوب **after()** يكمن في هيكل حلقة الرسوم المتحركة (أي دالة أو أي أسلوب "متكرر"، هذا يعني أن يطلق على نفسه) : ليس من السهل في الواقع إتقان هذا النوع من البناء المنطقي، خاصة إذا كنت تريد تعيين رسوم متحركة من عدة كائنات رسم مستقلة، بما في ذلك عدد الحركات التي ينبغي أن تختلف مع مرور الوقت .

تأخير الرسوم المتحركة باستخدام ()time.sleep

يمكنك تجاهل هذه القيود المفروضة على أسلوب **()after** المذكور أعلاه، إذا كنت تعطي الرسوم المتحركة الخاصة بكائنات رسومية للمواضيع المستقلة . بذلك، تقوم بتحرر وصاية **()mainloop**, ومن ثم يسمح لك ببناء إجراءات رسوم متحركة على أساس هياكل حلقات أكثر "كلاسيكية"، على سبيل المثال باستخدام عبارة **while** أو عبارة **for** .

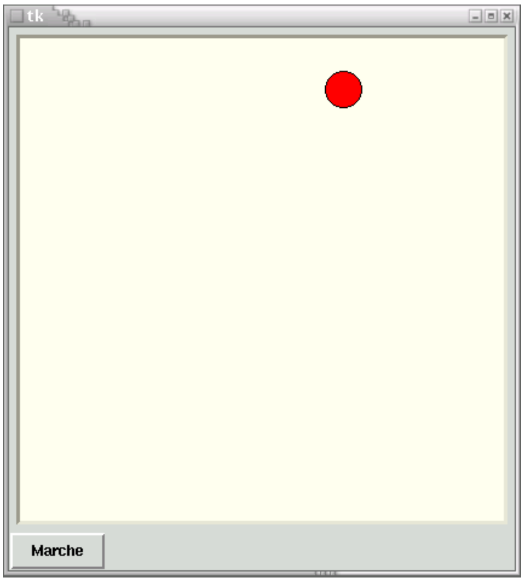
في قلب كل من هذه الحلقات، يجب عليك أن تكون دائما على يقين من أن إدراج تأخير أثناء "السير جانب" نظام التشغيل (حتى تتمكن من التعامل مع المواضيع الأخرى) . للقيام بذلك، سوف تستخدك الدالة **()sleep** من وحدة **time**. هذه الدالة تسمح لك بتعليق تنفيذ الموضوع الحالي لفترة معينة من الزمن، أثنائها المواضيع والتطبيقات الأخرى تستمر في العمل . إن التأخير المنتج على هذا النحو لا يعتمد على **()mainloop**, وبالتالي، يمكن أن يكون أقصر بكثير مما كنت تأذن أسلوب **()after**.

تنبيه : هذا لا يعني أن الشاشة سوف تجدد نفسها بشكل أسرع، لأن هذا التجديد لايزال يقدم من خلال **()mainloop**. يمكنك تسريعها إلى حد كبير من آليات مختلفة التي تصنعها بنفسك في إجراءات الرسوم المتحركة الخاصة بك . على سبيل المثال، في برنامج لعبة، فإنه من الشائع أن تكون المقارنة بين مواقع اثنين من الجوالات بشكل دوري (مثل قذيفة والهدف)، من أجل اتخاذ إجراءات عند انضمامهم (إنفجار، إضافة نقاط إلى النتيجة، إلخ) . باستخدام تقنية الرسوم المتحركة التي تم شرحها هنا، يمكنك أن تفعل الكثير في الكثير من الأحيان من هذه المقارنات، ونتوقع بالتالي نتائج أكثر دقة . وبالمثل، يمكنك زيادة عدد النقاط التي تؤخذ بعين الاعتبار لحساب المسار في الوقت الحقيقي، وبالتالي فإنه يتحسن .

> تنبيه
>
> عندما تستخدم الأسلوب ***()after***, يجب عليك أن تقوم بتحديد التأخير بالميلي ثانية. كبرامتر صحيح . عندما تستدعي الدالة ***()sleep***, البرامتر التي تقوم بتمريرها يجب أن تكون بالثواني. بشكل عدد حقيقي (*float*). ويمكنك استخدام أعداد صغيرة جدا (على سبيل المثال : 0.0003) .

مثال ملموس

السكريبت الصغير المستنسخ أدناه يوضح تنفيذ هذه التقنية في مثال بسيط عمدا . هو تطبيق رسومي صغير فيه تتحرك في دائرة داخل اللوحة. يتم تشغيل "محركه" **()mainloop** كالعادة على الموضوع الرئيسي . منشئ تطبيق المثيل لوحة تحتوي على رسم دائرة، وزر وكائن موضوع . هذا كائن الموضوع هو رسوم متحركة للرسم، لكن دون اللجوء إلى استدعاء الأسلوب **()after** للويدجت . فهو يستخدم حلقة بسيطة **while** كلاسيكية جدا، مثبة في أسلوبه **()run**.


tk
Marche

```
1#  from tkinter import *
2#  from math import sin, cos
3#  import time, threading
4#
5#  class App(Frame):
6#      def __init__(self):
7#          Frame.__init__(self)
8#          self.pack()
9#          can =Canvas(self, width =400, height =400,
10#                     bg ='ivory', bd =3, relief =SUNKEN)
11#         can.pack(padx =5, pady =5)
12#         cercle = can.create_oval(185, 355, 215, 385, fill ='red')
13#         tb = Thread_balle(can, cercle)
14#         Button(self, text ='Marche', command =tb.start).pack(side =LEFT)
15#         # Button(self, text ='Arrêt', command =tb.stop).pack(side =RIGHT)
16#         # إيقاف الخيط الاخر إذا قمنا بإغلاق النافذة :
17#         self.bind('<Destroy>', tb.stop)
18#
19# class Thread_balle(threading.Thread):
20#     def __init__(self, canevas, dessin):
21#         threading.Thread.__init__(self)
22#         self.can, self.dessin = canevas, dessin
23#         self.anim =1
24#
25#     def run(self):
26#         a = 0.0
27#         while self.anim == 1:
28#             a += .01
29#             x, y = 200 + 170*sin(a), 200 +170*cos(a)
30#             self.can.coords(self.dessin, x-15, y-15, x+15, y+15)
31#             time.sleep(0.010)
32#
33#     def stop(self, evt =0):
34#         self.anim =0
35#
36# App().mainloop()
```

تعليقات

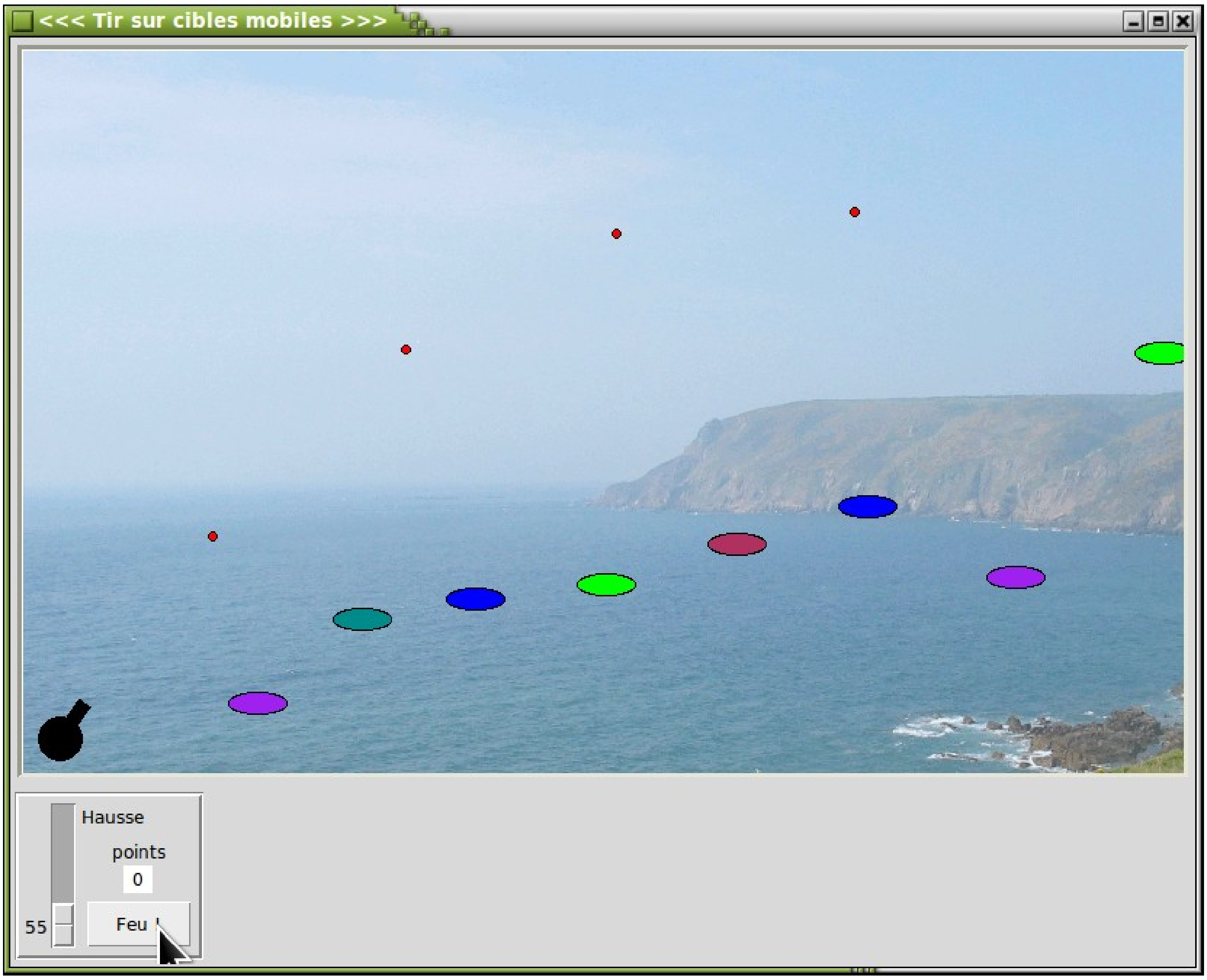
- السطران 13 و 14 : لتبسيط المثال إلى الحد الأقصى قمنا بإنشاء كائن موضوع مسؤول عن حركة الرسوم المتحركة، مباشرة في منشئ التطبيق الرئيسي . هذا كائن الموضوع سوف يتم تشغيله عندما يقوم المستخدم بالضغط على <Marche>، التي تقوم بتفعيل أسلوبه **start()** (تذكر هنا أن هذا الأسلوب المدمج الذي سيتم تشغيله هو نفس **run()** تذكر هنا أن هذا الأسلوب المدمج الذي سيتم تشغيله هو نفس run() حين أنشأنا حلقة الرسوم المتحركة الخاصة بنا) .
- السطر 15 : لا يمكن إعادة تشغيل موضوع انتهى . ولذلك، لا يمكنك تشغيل هذه الرسوم المتحركة سوى مرة واحدة فقط (على الأقل في شكل مقدمة هنا) . لإقناعك بذلك، فعل السطر 15 عن طريق إزالة الرمز **#** في البداية (و الذي تعتبره بيثون مجرد تعليق) : عند بدء الرسوم المتحركة، النقر بواسطة زر الفأرة يتسبب في استدعاء الخروج من حلقة **while** في الأسطر من 27 إلى 31، سوف ينهيها الأسلوب **run()**. ستتوقف الرسوم المتحركة، لكن الموضوع سوف ينتهي أيضا . إذا حاولت تشغيله باستخدام الزر <Marche>، لن تحصل سوى على رسالة خطأ .

• الأسطر من 26 إلى 31 : لمحاكات حركة دائرية موحدة، يكفي أن تغير باستمرار قيمة الزاوية **a**. الجيب وجيب التمام لهذه الزاوية يمكنك حساب إحداثيات **x** و **y** من نقطة محيط الدائرة التي تتطابق مع هذه الزاوية[113].279

في كل تكرار، سوف تتغير الزاوية بمئة راديان فقط (حوالي 0.6 درجة)، وسوف يتطلب ذلك 638 تكرار ليقوم بدورة كاملة . التوقيت الذي تم اختياره لهذه التكرارات هو في السطر 31 : 10 ميلي ثانية . يمكنك تسريع هذه العملية عن طريق تقليل هذه القيمة، ولكنك لا يمكنك أن تقلل أقل من 1 ميلي ثانية (0.001 ثانية)، وهي ليست سيئة للغاية .

> تذكير
>
> يمكنك الحصول على شفرة جميع الأمثلة من موقع :
>
> *http://inforef.be/swi/python.htm*, أو :
>
> *http://main.pythomium.net/download/cours_python.zip*
>
> سوف تجد في ملف يسمى ***cibles_multiples.py***, لعبة صغيرة يجب على المستخدم إطلاق النار على مجموعة من الأهداف المتحركة التي تزداد أعدادها وسرعتها مع مرور الوقت . وهذه اللعبة تستخدم تقنيات الرسوم المتحركة الموضحة أعلاه

113 يمكنك أن تجد بعض التفسيرات حول هذا الموضوع في صفحة 279.



<<< Tir sur cibles mobiles >>>
Hausse
points
0
55
Feu !

الملحق أ

تثبيت بيثون

إذا أردت تجربة بيثون على حاسوبك الخاص، لا تتردد : عملية التثبيت سهلة للغاية، بدون أي خطر على حاسوبك .

في نظام تشغيل ويندوز

على الموقع الرسمي لبيثون : http://www.python.org, سوف تجد قسم تحميل برامج التثبيت التلقائي لمختلف إصدارات بيثون . يمكنك اختيار خيار آخر "منتج" .

على سبيل المثال، في 2 ديسمبر (كانون الأول) 2011، تم إصدار الإصدار 3.2.2 . ملف التثبيت يسمى Python 3.2.2 Windows x86 MSI Installer (أو النسخة المطابقة لأنظمة 64 بت) . تحتوي هذه الإصدارة على المكتبة الرسومية tkinter .

انسخ هذا الملف إلى دليل مؤقت على جهاز الحاسوب الخاص بك. وقم بتشغيله . بيثون 3 سيتم تثبيته افتراضيا في دليل باسم **Python** (حيث ** أول رقمين من رقم الإصدار)، وسيتم تحديث أيقونات بيثون تلقائيا .

إذا كنت تريد الاستفادة من موارد مكتبات الطرف ثالث التي لم تتوفر بعد لبيثون 3، مثل ReportLab أو Python Imaging Library، يمكنك تثبيت مثبت الإصدار الأخير من بيثون 2، بتحميل الملف Python 2.7.2 Windows Installer (أو النسخة المخصصة لأجهزة 64بت) . يتم تثبيت النسختين في أدلة مختلفة دون أن يعوق أحدهم الآخر بأي شكل من الأشكال .

عند اكتمال التثبيت، يمكنك حذف الدليل المؤقت .

في نظام تشغيل لينكس

ربما قد قمت بتثبيت نظام تشغيل لينكس بمساعدة إحدى التوزيعات مثل أبنتو، سوزي، ريدهات ... قم ببساطة بتثبيت بيثون والتي هي جزء منه، ولا تستغنى عن tkinter (أيضا قم بتثبيته في نفس الوقت مع مكتبة Python Imaging Library) . إذا قمت بتثبيت بيثون 2 أو بيثون 3، يجب عليك أيضا تثبيت إصدارين مختلفين من tkinter .

في نظام تشغيل ماك

على الموقع الرسمي لبيثون، سوف تجد مجموعة تركيب لنظام ماك مثل التي لدى نظام تشغيل ويندوز .

تثبيت Cherrypy

مكتبة Cherrypy هي منتج مستقل له موقعه الرسمي على الإنترنت :

http://cherrypy.org. قم بزيارة قسم التنزيل :

http://download.cherrypy.org/cherrypy/3.2.2/

في هذا الدليل، سوف تجد روابط تحميل الإصدار الحالي من Cherrypy . في وقت كتابة هذه السطور (2011\12\02)، توجد نسخة 3.2.2 (هذا الرقم ليس له أي علاقة بإصدارات بيثون) . الملفات تتضمن نسخ Cherrypy لبيثون 2 وبيثون 3 في نفس المجموعة .

- الملف CherryPy-3.2.2.win32.exe، ثم قم بتشغيل الملف الذي قمت بتحميله (المثبت التلقائي) . إذا كنت قد ثبت نسختين بيثون أي الإصدار 2 و 3، فيجب عليك في هذه الحالة إعادة تشغيل المثبت مرتين لكي يثبت المكتبات في كلا الإصدارين .
- إذا كنت تعمل على نظام تشغيل لينكس أو أي نظام تشغيل أخر، فيجب عليك نقل ملف الأرشيف الذي حملته (CherryPy-3.2.2.tar.gz أو CherryPy-3.2.2.zip) إلى أي دليل مؤقت، ثم قم بفك ضغطه بمساعدة برنامج مناسب (unzip أو tar) . الفك يكشف عن دليل فرعي يسمى CherryPy-3.2.2 . قم بالدخول إلى هذا الدليل وأنت في وضع مستخدم الجذر (روت)، وقم بتنفيذ هذا الأمر: `python3 setup.py install` لتثبيت نسخة Cherrypy الخاصة ببيثون 3، أو\و الأمر: `python setup.py install` لتثبيت نسخة Cherrypy الخاصة ببيثون 2 .

تثبيت pg8000

مكتبة pg8000 هي واحدة من وحدات الواجهة التي تمكنك من الوصول إلى الخادم PostgreSQL من بيثون . وهي ليست الأكثر كفاءة، لكنها لديها ميزة أنها متاحة لبيثون 2 و 3 في وقت كتابة هذه الأسطر، وهذه الميزة لا توجد في كل المكتبات . بالإضافة إلى ذلك، هذه الوحدة مكتوبة ببيثون ولا تحتاج إلى أية مكتبة أخرى، بحيث تطبيقات بيثون التي تستخدمها تكون محمولة .

عندما تقرأ هذه الأسطر، إن الوحدات الأكثر كفاءة بالتأكيد متاحة، مثل psycopg2 . أرجو منك الإطلاع على مواقع الويب التي تتعامل مع تفاعل بيثون-PostgreSQL للمزيد إذا كنت ترغب في تطوير تطبيق مهم .

لتثبيت pg8000 على نظامك، قم بزيارة الموقع /http://pybrary.net/pg8000, وقم بتحميل الملف المناسب لأحدث نسخة متاحة، وهي مخصصة لبيثون 2 أو بيثون 3 (يمكنك مرة أخرى تثبيت النسختين) . ملف التحميل هو نفسه، سواء أن كنت تعمل على ويندوز أو لينكس أو ماك أو أي نظام تشغيل أخر، ويمكنك تحميله على شكل .`zip` أو .`tar.gz` (على سبيل المثال pour-8000 py3-1.08.tar.gz لبيثون 3 أو pg8000-1.08.tar.gz لبيثون 2 في وقت كتابة هذه الأسطر) . ثم قم بنسخ ملف الأرشيف الذي حملته إلى أي دليل مؤقت، ثم قم بفك ضغطه بمساعدة أي برنامج مناسب (unzip أو tar) . عملية الفك تكشف عن دليل فرعي pg8000-1.08 أو pg8000-py3-1.08 . قم بالدخول إلى هذا الدليل كمدير، واكتب الأمر : `python3 setup.py install` لتثبيت pg8000 المخصصة لبيثون 3، وأو الأمر: `python setup.py install` لتثبيت النسخة المخصصة لبيثون 2 .

تثبيت ReportLab و Python Imaging Library

في وقت كتابة هذه السطور، هذه المكتبة غير متاحة للأسف لبيثون 3 (انظر للفصل 18 للاطلاع على مناقشة هذه المشكلة) . لذا يجب عليك تثبيت واحدة من الإصدارات المناسبة لبيثون 2.6 وبيثون 2.7 .

لتثبيت ReportLab :

- في لينكس، يكفي أن تقوم بتثبيت حزمة **python-reportlab** المناسبة لتوزيعتك (أبنتو، ديبان ...) . للحصول على أخر إصدار، قم بالحصول عليها من شبكة الإنترنت، حزم ReportLab موجودة على العنوان التالي: .http://www.reportlab.com/ftp
- في نظام تشغيل ويندوز، اعتمد على نسخة بيثون التي لديك (2.6 أو 2.7)، قم بتحميل الملف **reportlab-2.5.win32-py2.6.exe** أو **reportlab-2.5.win32-py2.7.exe**, ثم قم بتشغيله (التثبيت التلقائي) .
- بالنسبة للأنظمة التشغيل الأخرى، قم بتحميل إحدى ملفات الأرشيف **reportlab-2.5.tar.gz** أو **reportlab-2.5.zip** , وقم بفك ضغطها في دليل مؤقت . سيتم إنشاء دليل فرعي تلقائيا، قم بتنفيذ هذا الأمر `python setup.py install` وأنت مدير .

لتثبيت Python Imaging :

- في لينكس، يكفي أن تقوم بتثبيت حزمة **python-imaging** المناسبة لتوزيعتك (أبنتو، ديبان ...) . للحصول على أخر إصدار، قم بالحصول عليها من شبكة الإنترنت، حزم Python Imaging Library موجودة على العنوان التالي: ./http://www.pythonware.com/products/pil

- في نظام تشغيل ويندوز، قم بتحميل الملف **Python Imaging Library 1.1.7 for Python 2.6** أو **Python Imaging Library 1.1.7 for Python 2.7** لبيثون 2.6 أو لبيثون 2.7، ثم قم بتشغيله (التثبيت التلقائي) .
- بالنسبة لأنظمة التشغيل الأخرى، قم بتحميل **Python Imaging Library 1.1.7 Source Kit** (جميع *الأنظمة*), وقم بفك ضغطها في دليل مؤقت . سيتم إنشاء دليل فرعي تلقائيا، قم بتنفيذ هذا الأمر `python setup.py install` وأنت مدير .

الملحق ب

حلول التمارين

لبعض التمارين، لن نوفر لك الحل. حاول أن تقوم بالعثور على الحل بدون مساعدة، حتى لو كان هذا يبدوا صعبًا. و هذا في الواقع يجعلك تثابر لتحل مثل هذه المشاكل حتى تتعلم أفضل.

التمرين 4.2 :

```
>>> c = 0
>>> while c < 20:
...     c = c +1
...     print(c, "x 7 =", c*7)
```

أو :

```
>>> c = 1
>>> while c <= 20:
...     print(c, "x 7 =", c*7)
...     c = c +1
```

التمرين 4.3 :

```
>>> s = 1
>>> while s <= 16384:
...     print(s, "euro(s) =", s *1.65, "dollar(s)")
...     s = s *2
```

التمرين 4.4 :

```
>>> a, c = 1, 1
>>> while c < 13:
...     print(a, end =' ')
...     a, c = a *3, c+1
```

التمرين 4.6 :

```
# Le nombre de secondes est fourni au départ :
# (مطلوب عدد كبير!)
nsd = 12345678912
```

```
# عدد الثواني في يوم واحد:
nspj = 3600 * 24
# عدد الثواني في السنة (365 يوما -
# لا تأخذ في الاعتبار السنوات الكبيسة):
nspa = nspj * 365
# عدد الثواني في الشهر (على افتراض
# أن كل شهر به 30 يوما):
nspm = nspj * 30
# Nombre d'années contenues dans la durée fournie :
na = nsd // nspa        # division <entière>
nsr = nsd % nspa        # n. de sec. restantes
# عدد الأشهر المتبقية:
nmo = nsr // nspm       # division <entière>
nsr = nsr % nspm        # n. de sec. restantes
# عدد الأيام المتبقية:
nj = nsr // nspj        # division <entière>
nsr = nsr % nspj        # n. de sec. restantes
# عدد الساعات المتبقية:
nh = nsr // 3600        # division <entière>
nsr = nsr % 3600        # n. de sec. restantes
# عدد الدقائق المتبقية:
nmi = nsr // 60         # division <entière>
nsr = nsr % 60          # n. de sec. restantes

print("Nombre de secondes à convertir :", nsd)
print("Cette durée correspond à", na, "années de 365 jours, plus")
print(nmo, "mois de 30 jours,", end=' ')
print(nj, "jours,", end=' ')
print(nh, "heures,", end=' ')
print(nmi, "minutes et", end=' ')
print(nsr, "secondes.")
```

التمرين 4.7 :

```
# affichage des 20 premiers termes de la table par 7,
# avec signalement des multiples de 3 :

i = 1               # compteur : prendra successivement les valeurs de 1 à 20
while i < 21:
    # calcul du terme à afficher :
    t = i * 7
    # affichage sans saut à la ligne (utilisation de la virgule) :
    print(t, end =’ ‘)
    # ce terme est-il un multiple de 3 ? (utilisation de l'opérateur modulo) :
    if t % 3 == 0:
        print("*", end =’ ‘)    # affichage d'une astérisque dans ce cas
    i = i + 1                   # incrémentation du compteur dans tous les cas
```

التمرين 5.1 :

```
# Conversion degrés -> radians
# Rappel : un angle de 1 radian est un angle qui correspond à une portion
# de circonférence de longueur égale à celle du rayon.
# Puisque la circonférence vaut 2 pi R, un angle de 1 radian correspond
# à 360° / 2 pi , ou encore à 180° / pi
```

```
# Angle fourni au départ en degrés, minutes, secondes :
deg, min, sec  = 32, 13, 49

# Conversion des secondes en une fraction de minute :
fm = sec/60
# Conversion des minutes en une fraction de degré :
fd = (min + fm)/60
# Valeur de l'angle en degrés "décimalisés" :
ang = deg + fd
# Valeur de pi :
pi = 3.14159265359
# Valeur d'un radian en degrés :
rad = 180 / pi
# Conversion de l'angle en radians :
arad = ang / rad
# Affichage :
print(deg, "°", min, "'", sec, '" =', arad, "radian(s)")
```

التمرين 5.3 :

```
# تحويل °الفهرنهايت <-> °السيليزي

# أ) درجة الحرارة بـ °س :
tempC = 25
# التحويل إلى °فهرنهايت :
tempF = tempC * 1.8 + 32
# عرض :
print(tempC, "°C =", tempF, "°F")

# ب) درجة الحرارة بـ °ف :
tempF = 25
# التحويل إلى °سيليزي :
tempC = (tempF - 32) / 1.8
# عرض :
print(tempF, "°F =", tempC, "°C")
```

التمرين 5.5 :

```
n = 1       # عدد المربعات
g = 1       # nombre de grains à y déposer
# Pour la variante, il suffit de définir g comme <float>
# en remplaçant la ligne ci-dessus par :  g = 1.

while n < 65 :
    print(n, g)
    n, g = n+1, g*2
```

التمرين 5.6 :

```
# البحث عن حرف معين في السلسلة

# السلسلة المعطاة:
ch = "Monty python flying circus"
# مفتاح البحث:
cr = "e"
```

```
#البحث الصحيح:
lc = len(ch)     # عدد الأحرف التي سيتم اختبارها
i = 0            # indice du caractère en cours d'examen
t = 0            # "drapeau" à lever si le caractère recherché est présent
while i < lc:
    if ch[i] == cr:
        t = 1
    i = i + 1
# عرض:
print("Le caractère", cr, end =' ')
if t == 1:
    print("est présent", end =' ')
else:
    print("est inrouvable", end =' ')
print("dans la chaîne", ch)
```

التمرين 5.8 :

```
# إدراج حرف مسافة في سلسلة

# السلسلة المعطاة:
ch = "Véronique"
# إدراج الحرف:
cr = "*"
# Le nombre de caractères à insérer est inférieur d'une unité au
# nombre de caractères de la chaîne. On traitera donc celle-ci à
# partir de son second caractère (en omettant le premier).
lc = len(ch)     # إجمالي عدد الأحرف
i = 1            # indice du premier caractère à examiner (le second, en fait)
nch = ch[0]      # nouvelle chaîne à construire (contient déjà le premier car.)
while i < lc:
    nch = nch + cr + ch[i]
    i = i + 1
# عرض:
print(nch)
```

التمرين 5.9 :

```
# عكس سلسلة أحرف #

# السلسلة المعطاة:
ch = "zorglub"
lc = len(ch)       # عدد الأحرف الإجمالي
i = lc - 1         # البدأ من الحرف الأخير
nch = ""           # لبدأ سلسلة جديدة فارغة
while i >= 0:
    nch = nch + ch[i]
    i = i - 1
# عرض:
print(nch)
```

التمرين 5.11 :

```
# مزج قائمتين في واحدة
```

```
# القوائم المعطاة :
t1 = [31,28,31,30,31,30,31,31,30,31,30,31]
t2 = ['Janvier','Février','Mars','Avril','Mai','Juin',
      'Juillet','Août','Septembre','Octobre','Novembre','Décembre']
# عمل قائمة جديدة فارغة:
t3 = []
# تجهيز حلقة :
i = 0
while i < len(t1):
    t3.append(t2[i])
    t3.append(t1[i])
    i = i + 1

# عرض:
print(t3)
```

التمرين 5.12 :

```
# عرض العناصر في قائمة

# القائمة المعطاة:
t2 = ['Janvier','Février','Mars','Avril','Mai','Juin',
      'Juillet','Août','Septembre','Octobre','Novembre','Décembre']
# عرض:
i = 0
while i < len(t2):
    print(t2[i], end =' ')
    i = i + 1
```

التمرين 5.13 :

```
# البحث عن أكبر عدد في قائمة

# القائمة المعطاة :
tt = [32, 5, 12, 8, 3, 75, 2, 15]
# Au fur et à mesure du traitement de la liste, on mémorisera dans
# la variable ci-dessous la valeur du plus grand élément déjà trouvé :
max = 0
# استعراض لجميع العناصر :
i = 0
while i < len(tt):
    if tt[i] > max:
        max = tt[i]         # تخزين أكبر عدد الجديد
    i = i + 1
# عرض:
print("Le plus grand élément de cette liste a la valeur", max)
```

التمرين 5.14 :

```
# فصل الأعداد الفردية والزوجية

# القوائم المعطاة :
tt = [32, 5, 12, 8, 3, 75, 2, 15]
```

```
pairs = []
impairs = []
# استعراض لجميع العناصر :
i = 0
while i < len(tt):
    if tt[i] % 2 == 0:
        pairs.append(tt[i])
    else:
        impairs.append(tt[i])
    i = i + 1
# عرض:
print("Nombres pairs :", pairs)
print("Nombres impairs :", impairs)
```

التمرين 6.1 :

```
# تحويل ميل/ساعة إلى كم/ساعة و متر/ثانية

print("Veuillez entrer le nombre de miles parcourus en une heure : ", end =' ')
ch = input()
mph = float(ch)             # تحويل سلسلة لرقم حقيقي
mps = mph * 1609 / 3600     # التحويل لمتر في الثانية
kmph = mph * 1.609          # التحويل لكيلومتر في الساعة
# عرض:
print(mph, "miles/heure =", kmph, "km/h, ou encore", mps, "m/s")
```

التمرين 6.2 :

```
# محيط ومساحة أي مثلث

from math import sqrt

print("Veuillez entrer le côté a : ")
a = float(input())
print("Veuillez entrer le côté b : ")
b = float(input())
print("Veuillez entrer le côté c : ")
c = float(input())
d = (a + b + c)/2                # نصف المحيط
s = sqrt(d*(d-a)*(d-b)*(d-c))    # المساحة (اعتمادا على معادلة)

print("Longueur des côtés =", a, b, c)
print("Périmètre =", d*2, "Aire =", s)
```

التمرين 6.4 :

```
# إدخال عناصر في قائمة

tt = []             # القائمة الكاملة فارغة
ch = "start"        # أي قيمة (ما عدا اللاشيء)
while ch != "":
    print("Veuillez entrer une valeur : ")
    ch = input()
    if ch != "":
```

```
        tt.append(float(ch))        # مختلف عن : tt.append(ch)

# عرض القائمة:
print(tt)
```

التمرين 6.8 :

```
# Traitement de nombres entiers compris entre deux limites

print("Veuillez entrer la limite inférieure :", end=' ')
a = eval(input())
print("Veuillez entrer la limite supérieure :", end=' ')
b = eval(input())
s = 0                   # somme recherchée (nulle au départ)
# Parcours de la série des nombres compris entre a et b :
n = a                   # nombre en cours de traitement
while n <= b:
    if n % 3 ==0 and n % 5 ==0:      # variante : 'or' au lieu de 'and'
        s = s + n
    n = n + 1

print("La somme recherchée vaut", s)
```

التمرين 6.9 :

```
# السنوات الكبيسة

print("Veuillez entrer l'année à tester :", end=' ')
a = eval(input())

if a % 4 != 0:
    # a لا يُقسم على 4 -> السنة بسيطة
    bs = 0
else:
    if a % 400 ==0:
        # a يُقسم على 400 -> السنة كبيسة
        bs = 1
    elif a % 100 ==0:
        # a يُقسم على 100 -> السنة بسيطة
        bs = 0
    else:
        # باقي حالات القسمة على 4 -> السنة كبيسة
        bs = 1
if bs ==1:
    ch = "est"
else:
    ch = "n'est pas"
print("L'année", a, ch, "bissextile.")

########## بديل (اقترحه أليكس مصباح) : #####

a = eval(input('Veuillez entrer une année :'))

if (a%4==0) and ((a%100!=0) or (a%400==0)):
    print(a,"est une année bissextile")
else:
    print(a,"n'est pas une année bissextile")
```

التمرين 6.11 : حسابات المثلثات

```
from sys import exit      # وحدة تحتوي على وظائف النظام

print("""
Veuillez entrer les longueurs des 3 côtés
(en séparant ces valeurs à l'aide de virgules) :""")
a, b, c = eval(input())
# Il n'est possible de construire un triangle que si chaque côté
# a une longueur inférieure à la somme des deux autres :
if a < (b+c) and b < (a+c) and c < (a+b) :
    print("Ces trois longueurs déterminent bien un triangle.")
else:
    print("Il est impossible de construire un tel triangle !")
    exit()          # ainsi l'on n'ira pas plus loin.

f = 0
if a == b and b == c :
    print("Ce triangle est équilatéral.")
    f = 1
elif a == b or b == c or c == a :
    print("Ce triangle est isocèle.")
    f = 1
if a*a + b*b == c*c or b*b + c*c == a*a or c*c + a*a == b*b :
    print("Ce triangle est rectangle.")
    f = 1
if f == 0 :
    print("Ce triangle est quelconque.")
```

التمرين 6.15 :

```
# ملاحظات الواجبات المدرسية

notes = []           # عمل قائمة
n = 2                # أي قيمة موجبة لبدء الحلقة
while n >= 0 :
    print("Entrez la note suivante, s.v.p. : ", end=' ')
    n = float(input())          # تحويل الإدخال لعدد حقيقي
    if n < 0 :
        print("OK. Terminé.")
    else:
        notes.append(n)         # إضافة ملاحظة للقائمة
        # Calculs divers sur les notes déjà entrées :
        # valeurs minimale et maximale + total de toutes les notes.
        min = 500               # valeur supérieure à toute note
        max, tot, i = 0, 0, 0
        nn = len(notes)         # nombre de notes déjà entrées
        while i < nn:
            if notes[i] > max:
                max = notes[i]
            if notes[i] < min:
                min = notes[i]
            tot = tot + notes[i]
            moy = tot/nn
            i = i + 1
        print(nn, "notes entrées. Max =", max, "Min =", min, "Moy =", moy)
```

التمرين 7.3 :

```
from math import pi

def surfCercle(r):
    "Surface d'un cercle de rayon r"
    return pi * r**2

# التجربة:
print(surfCercle(2.5))
```

التمرين 7.4 :

```
def volBoite(x1, x2, x3):
    "Volume d'une boîte parallélipipédique"
    return x1 * x2 * x3

# التجربة:
print(volBoite(5.2, 7.7, 3.3))
```

التمرين 7.5 :

```
def maximum(n1, n2, n3):
    "Renvoie le plus grand de trois nombres"
    if n1 >= n2 and n1 >= n3:
        return n1
    elif n2 >= n1 and n2 >= n3:
        return n2
    else:
        return n3

# التجربة:
print(maximum(4.5, 5.7, 3.9))
print(maximum(8.2, 2.1, 6.7))
print(maximum(1.3, 4.8, 7.6))
```

التمرين 7.9 :

```
def compteCar(ca, ch):
    "Renvoie le nombre de caractères ca trouvés dans la chaîne ch"
    i, tot = 0, 0
    while i < len(ch):
        if ch[i] == ca:
            tot = tot + 1
        i = i + 1
    return tot

# التجربة:
print(compteCar("e","Cette chaîne est un exemple"))
```

التمرين 7.10 :

```
def indexMax(tt):
    "renvoie l'indice du plus grand élément de la liste tt"
```

```
    i, max = 0, 0
    while i < len(tt):
        if tt[i] > max :
            max, imax = tt[i], i
        i = i + 1
    return imax

# التجربة:
serie = [5, 8, 2, 1, 9, 3, 6, 4]
print(indexMax(serie))
```

التمرين 7.11 :

```
def nomMois(n):
    "renvoie le nom du n-ième mois de l'année"
    mois = ['Janvier,', 'Février', 'Mars', 'Avril', 'Mai', 'Juin', 'Juillet',
            'Août', 'Septembre', 'Octobre', 'Novembre', 'Décembre']
    return mois[n -1]       # les indices sont numérotés à partir de zéro

# التجربة:
print(nomMois(4))
```

التمرين 7.14 :

```
def volBoite(x1 =10, x2 =10, x3 =10):
    "Volume d'une boîte parallélipipédique"
    return x1 * x2 * x3

# التجربة:
print(volBoite())
print(volBoite(5.2))
print(volBoite(5.2, 3))
```

التمرين 7.15 :

```
def volBoite(x1 =-1, x2 =-1, x3 =-1):
    "Volume d'une boîte parallélipipédique"
    if x1 == -1 :
        return x1           # aucun argument n'a été fourni
    elif x2 == -1 :
        return x1**3        # un seul argument -> boîte cubique
    elif x3 == -1 :
        return x1*x1*x2     # deux arguments -> boîte prismatique
    else :
        return x1*x2*x3

# التجربة:
print(volBoite())
print(volBoite(5.2))
print(volBoite(5.2, 3))
print(volBoite(5.2, 3, 7.4))
```

التمرين 7.16 :

```
def changeCar(ch, ca1, ca2, debut =0, fin =-1):
    "Remplace tous les caractères ca1 par des ca2 dans la chaîne ch"
```

```
    if fin == -1:
        fin = len(ch)
    nch, i = "", 0            # nch : nouvelle chaîne à construire
    while i < len(ch) :
        if i >= debut and i <= fin and ch[i] == ca1:
            nch = nch + ca2
        else :
            nch = nch + ch[i]
        i = i + 1
    return nch

# التجربة:
print((changeCar("Ceci est une toute petite phrase", " ", "*")))
print((changeCar("Ceci est une toute petite phrase", " ", "*", 8, 12)))
print((changeCar("Ceci est une toute petite phrase", " ", "*", 12)))
print((changeCar("Ceci est une toute petite phrase", " ", "*", fin =12)))
```

التمرين 7.17 :

```
def eleMax(lst, debut =0, fin =-1):
    "renvoie le plus grand élément de la liste lst"
    if fin == -1:
        fin = len(lst)
    max, i = 0, 0
    while i < len(lst):
        if i >= debut and i <= fin and lst[i] > max:
            max = lst[i]
        i = i + 1
    return max

# التجربة:
serie = [9, 3, 6, 1, 7, 5, 4, 8, 2]
print(eleMax(serie))
print(eleMax(serie, 2, 5))
print(eleMax(serie, 2))
print(eleMax(serie, fin =3, debut =1))
```

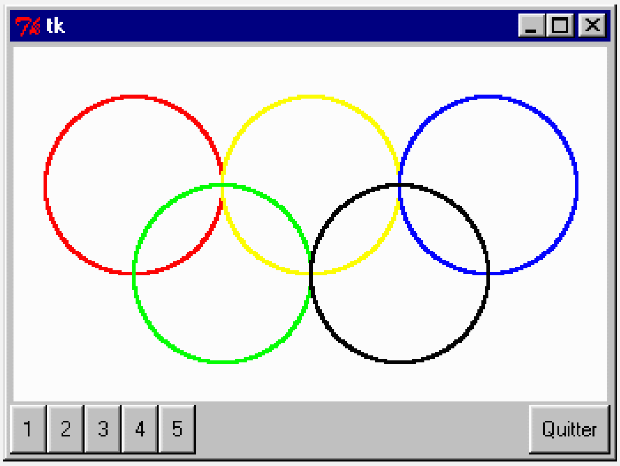
التمرين 8.7 :

```
from tkinter import *

# إحداثيات X,Y لـ 5 حلقات :
coord = [[20,30], [120,30], [220, 30], [70,80], [170,80]]
# ألوان الـ 5 حلقات :
coul = ["red", "yellow", "blue", "green", "black"]

base = Tk()
can = Canvas(base, width =335, height =200, bg ="white")
can.pack()
bou = Button(base, text ="Quitter", command =base.quit)
bou.pack(side = RIGHT)
# رسم الـ 5 حلقات :
i = 0
while i < 5:
    x1, y1 = coord[i][0], coord[i][1]
    can.create_oval(x1, y1, x1+100, y1 +100, width =2, outline =coul[i])
    i = i +1
base.mainloop()
```

للتنويع:




```
from tkinter import *

# رسم الـ 5 حلقات :
def dessineCercle(i):
    x1, y1 = coord[i][0], coord[i][1]
    can.create_oval(x1, y1, x1+100, y1 +100, width =2, outline =coul[i])

def a1():
    dessineCercle(0)

def a2():
    dessineCercle(1)

def a3():
    dessineCercle(2)

def a4():
    dessineCercle(3)

def a5():
    dessineCercle(4)

# إحداثيات X,Y لـ 5 حلقات :
coord = [[20,30], [120,30], [220, 30], [70,80], [170,80]]
# ألوان الـ 5 حلقات :
coul = ["red", "yellow", "blue", "green", "black"]

base = Tk()
can = Canvas(base, width =335, height =200, bg ="white")
can.pack()
bou = Button(base, text ="Quitter", command =base.quit)
bou.pack(side = RIGHT)

# تركيب 5 أزرار:
Button(base, text='1', command = a1).pack(side =LEFT)
Button(base, text='2', command = a2).pack(side =LEFT)
Button(base, text='3', command = a3).pack(side =LEFT)
Button(base, text='4', command = a4).pack(side =LEFT)
Button(base, text='5', command = a5).pack(side =LEFT)
base.mainloop()
```

التمرينان 8.9 و 8.10 :

```
# Dessin d'un damier, avec placement de pions au hasard

from tkinter import *
from random import randrange        # générateur de nombres aléatoires

def damier():
    "dessiner dix lignes de carrés avec décalage alterné"
    y = 0
    while y < 10:
        if y % 2 == 0:              # une fois sur deux, on
            x = 0                   # commencera la ligne de
        else:                       # carrés avec un décalage
            x = 1                   # de la taille d'un carré
        ligne_de_carres(x*c, y*c)
        y += 1

def ligne_de_carres(x, y):
    "dessiner une ligne de carrés, en partant de x, y"
    i = 0
    while i < 5:
        can.create_rectangle(x, y, x+c, y+c, fill='navy')
        i += 1
        x += c*2                    # espacer les carrés

def cercle(x, y, r, coul):
    "dessiner un cercle de centre x,y et de rayon r"
    can.create_oval(x-r, y-r, x+r, y+r, fill=coul)

def ajouter_pion():
    "dessiner un pion au hasard sur le damier"
    # tirer au hasard les coordonnées du pion :
    x = c/2 + randrange(10) * c
    y = c/2 + randrange(10) * c
    cercle(x, y, c/3, 'red')

##### Programme principal : ############

# Tâchez de bien "paramétrer" vos programmes, comme nous l'avons
# fait dans ce script. Celui-ci peut en effet tracer des damiers
# de n'importe quelle taille en changeant seulement la valeur
# d'une seule variable, à savoir la dimension des carrés :

c = 30                  # taille des carrés

fen = Tk()
can = Canvas(fen, width =c*10, height =c*10, bg ='ivory')
can.pack(side =TOP, padx =5, pady =5)
b1 = Button(fen, text ='damier', command =damier)
b1.pack(side =LEFT, padx =3, pady =3)
b2 = Button(fen, text ='pions', command =ajouter_pion)
b2.pack(side =RIGHT, padx =3, pady =3)
fen.mainloop()#
```

التمرين 8.12 :

```
# Simulation du phénomène de gravitation universelle
```

```
from tkinter import *
from math import sqrt

def distance(x1, y1, x2, y2):
    "distance séparant les points x1,y1 et x2,y2"
    d = sqrt((x2-x1)**2 + (y2-y1)**2)       # théorème de Pythagore
    return  d

def forceG(m1, m2, di):
    "force de gravitation s'exerçant entre m1 et m2 pour une distance di"
    return m1*m2*6.67e-11/di**2             # قانون نيوتن

def avance(n, gd, hb):
    "déplacement de l'astre n, de gauche à droite ou de haut en bas"
    global x, y, step
    # الإحداثيات الجديدة :
    x[n], y[n] = x[n] +gd, y[n] +hb
    # déplacement du dessin dans le canevas :
    can.coords(astre[n], x[n]-10, y[n]-10, x[n]+10, y[n]+10)
    # calcul de la nouvelle interdistance :
    di = distance(x[0], y[0], x[1], y[1])
    # conversion de la distance "écran" en distance "astronomique" :
    diA = di*1e9            # (1 pixel => 1 million de km)
    # calcul de la force de gravitation correspondante :
    f = forceG(m1, m2, diA)
    # affichage des nouvelles valeurs de distance et force :
    valDis.configure(text="Distance = " +str(diA) +" m")
    valFor.configure(text="Force = " +str(f) +" N")
    # adaptation du "pas" de déplacement en fonction de la distance :
    step = di/10

def gauche1():
    avance(0, -step, 0)

def droite1():
    avance(0, step, 0)

def haut1():
    avance(0, 0, -step)

def bas1():
    avance(0, 0, step)

def gauche2():
    avance(1, -step, 0)

def droite2():
    avance (1, step, 0)

def haut2():
    avance(1, 0, -step)

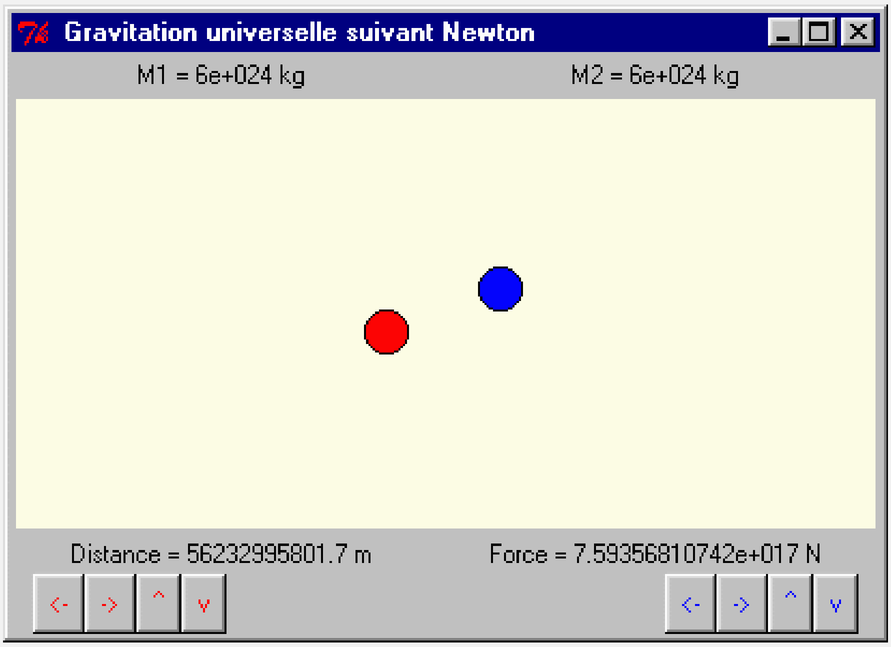
def bas2():
    avance(1, 0, step)

# Masses des deux astres :
m1 = 6e24          # (valeur de la masse de la terre, en kg)
m2 = 6e24          #
astre = [0]*2      # liste servant à mémoriser les références des dessins
x =[50., 350.]     # liste des coord. X de chaque astre (à l'écran)
y =[100., 100.]    # liste des coord. Y de chaque astre
```

```
step =10            # "pas" de déplacement initial

# Construction de la fenêtre :
fen = Tk()
fen.title(' Gravitation universelle suivant Newton')
# Libellés :
valM1 = Label(fen, text="M1 = " +str(m1) +" kg")
valM1.grid(row =1, column =0)
valM2 = Label(fen, text="M2 = " +str(m2) +" kg")
valM2.grid(row =1, column =1)
valDis = Label(fen, text="Distance")
valDis.grid(row =3, column =0)
valFor = Label(fen, text="Force")
valFor.grid(row =3, column =1)
# Canevas avec le dessin des 2 astres:
can = Canvas(fen, bg ="light yellow", width =400, height =200)
can.grid(row =2, column =0, columnspan =2)
astre[0] = can.create_oval(x[0]-10, y[0]-10, x[0]+10, y[0]+10,
                           fill ="red", width =1)
astre[1] = can.create_oval(x[1]-10, y[1]-10, x[1]+10, y[1]+10,
                           fill ="blue", width =1)
# 2 groupes de 4 boutons, chacun installé dans un cadre (frame) :
fra1 = Frame(fen)
fra1.grid(row =4, column =0, sticky =W, padx =10)
Button(fra1, text="<-", fg ='red',command =gauche1).pack(side =LEFT)
Button(fra1, text="->", fg ='red', command =droite1).pack(side =LEFT)
Button(fra1, text="^", fg ='red', command =haut1).pack(side =LEFT)
Button(fra1, text="v", fg ='red', command =bas1).pack(side =LEFT)
fra2 = Frame(fen)
fra2.grid(row =4, column =1, sticky =E, padx =10)
Button(fra2, text="<-", fg ='blue', command =gauche2).pack(side =LEFT)
Button(fra2, text="->", fg ='blue', command =droite2).pack(side =LEFT)
Button(fra2, text="^", fg ='blue', command =haut2).pack(side =LEFT)
Button(fra2, text="v", fg ='blue', command =bas2).pack(side =LEFT)

fen.mainloop()
```



التمرين 8.16 :

```
# تحويل درجات الحرارة فهرنهايت <=> سيليزي

from tkinter import *

def convFar(event):
    "valeur de cette température, exprimée en degrés Fahrenheit"
    tF = eval(champTC.get())
    varTF.set(str(tF*1.8 +32))

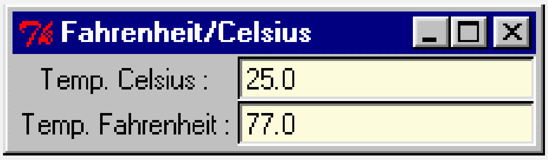
def convCel(event):
    "valeur de cette température, exprimée en degrés Celsius"
    tC = eval(champTF.get())
    varTC.set(str((tC-32)/1.8))

fen = Tk()
fen.title('Fahrenheit/Celsius')

Label(fen, text='Temp. Celsius :').grid(row =0, column =0)
# "variable tkinter" associée au champ d'entrée. Cet "objet-variable"
# assure l'interface entre TCL et Python (voir notes, page 165) :
varTC =StringVar()
champTC = Entry(fen, textvariable =varTC)
champTC.bind("<Return>", convFar)
champTC.grid(row =0, column =1)
# تهيئة محتويات متغير tkinter :
varTC.set("100.0")

Label(fen, text='Temp. Fahrenheit :').grid(row =1, column =0)
varTF =StringVar()
champTF = Entry(fen, textvariable =varTF)
champTF.bind("<Return>", convCel)
champTF.grid(row =1, column =1)
varTF.set("212.0")

fen.mainloop()
```



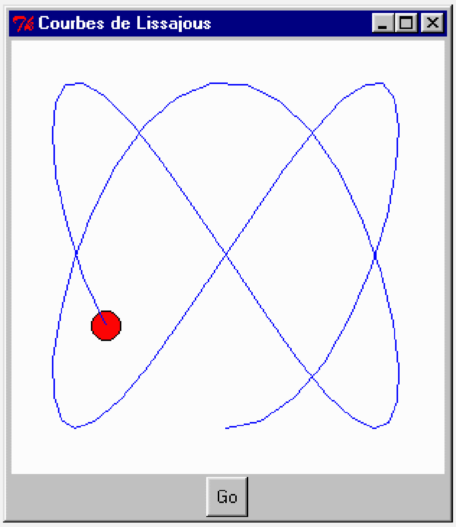
التمرينان 8.18 و 8.20 :

```
# Cercles et courbes de Lissajous

from tkinter import *
from math import sin, cos

def move():
    global ang, x, y
    # on mémorise les coordonnées précédentes avant de calculer les nouvelles :
    xp, yp = x, y
    # rotation d'un angle de 0.1 radian :
    ang = ang +.1
    # sinus et cosinus de cet angle => coord. d'un point du cercle trigono.
    x, y = sin(ang), cos(ang)
    # Variante déterminant une courbe de Lissajous avec f1/f2 = 2/3 :
    # x, y = sin(2*ang), cos(3*ang)
    # mise à l'échelle (120 = rayon du cercle, (150,150) = centre du canevas)
    x, y = x*120 + 150, y*120 + 150
    can.coords(balle, x-10, y-10, x+10, y+10)
    can.create_line(xp, yp, x, y, fill ="blue")     # تتبع المسار
```

```
ang, x, y = 0., 150., 270.
fen = Tk()
fen.title('Courbes de Lissajous')
can = Canvas(fen, width =300, height=300, bg="white")
can.pack()
balle = can.create_oval(x-10, y-10, x+10, y+10, fill='red')
Button(fen, text='Go', command =move).pack()
fen.mainloop()
```



التمرين 8.27 :

```
# سقوط وترتد

from tkinter import *

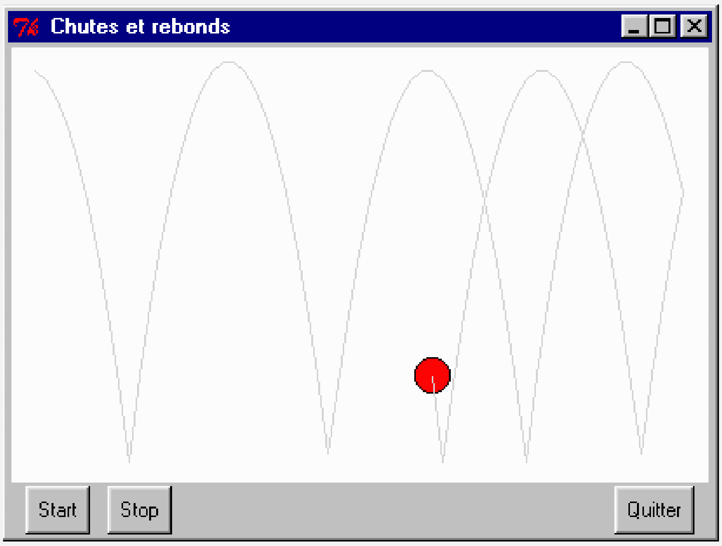
def move():
    global x, y, v, dx, dv, flag
    xp, yp = x, y              # mémorisation des coord. précédentes
    # حركة أفقية :
    if x > 385 or x < 15 :     # الارتداد على الجدران الجانبية:
        dx = -dx               # عكس الحركة
    x = x + dx
    # اختلاف السرعة العمودية (دائما للأسفل):
    v = v + dv
    # déplacement vertical (proportionnel à la vitesse)
    y = y + v
    if y > 240:                # مستوى سطح الأرض 240 بكسل :
        y = 240                #  défense d'aller + loin !
        v = -v                 # ارتداد: يتم عكس السرعة
    # إعادة الكرة:
    can.coords(balle, x-10, y-10, x+10, y+10)
    # رسم المسار:
    can.create_line(xp, yp, x, y, fill ='light grey')
    # ... et on remet ça jusqu'à plus soif :
    if flag > 0:
        fen.after(50,move)
```

```
def start():
    global flag
    flag = flag +1
    if flag == 1:
        move()

def stop():
    global flag
    flag =0


# إحداثيات التهيئة والسرعة والتحكم الرسوم المتحركة:
x, y, v, dx, dv, flag  = 15, 15, 0, 6, 5, 0

fen = Tk()
fen.title(' Chutes et rebonds')
can = Canvas(fen, width =400, height=250, bg="white")
can.pack()
balle = can.create_oval(x-10, y-10, x+10, y+10, fill='red')
Button(fen, text='Start', command =start).pack(side =LEFT, padx =10)
Button(fen, text='Stop', command =stop).pack(side =LEFT)
Button(fen, text='Quitter', command =fen.quit).pack(side =RIGHT, padx =10)
fen.mainloop()
```



التمرين 8.33 (لعبة الثعبان)

Nous ne fournissons ici qu'une première ébauche du script : le principe d'animation du « serpent ». Si le cœur vous en dit, vous pouvez continuer le développement pour en faire un véritable jeu, mais c'est du travail !

```
from tkinter import *

# === تحديد بعض معالجات الأحداث :

def start_it():
```

```
    "Démarrage de l'animation"
    global flag
    if flag ==0:
        flag =1
        move()

def stop_it():
    "Arrêt de l'animation"
    global flag
    flag =0

def go_left(event =None):
    "délacement vers la gauche"
    global dx, dy
    dx, dy = -1, 0

def go_right(event =None):
    global dx, dy
    dx, dy = 1, 0

def go_up(event =None):
    "déplacement vers le haut"
    global dx, dy
    dx, dy = 0, -1

def go_down(event =None):
    global dx, dy
    dx, dy = 0, 1

def move():
    "Animation du serpent par récursivité"
    global flag
    # Principe du mouvement opéré : on déplace le carré de queue, dont les
    # caractéristiques sont mémorisées dans le premier élément de la liste
    # <serp>, de manière à l'amener en avant du carré de tête, dont les
    # caractéristiques sont mémorisées dans le dernier élément de la liste.
    # On définit ainsi un nouveau carré de tête pour le serpent, dont on
    # mémorise les caractéristiques en les ajoutant à la liste.
    # Il ne reste plus qu'à effacer alors le premier élément de la liste,
    # et ainsi de suite ... :
    c = serp[0]             # extraction des infos concernant le carré de queue
    cq = c[0]               # réf. de ce carré (coordonnées inutiles ici)
    l =len(serp)            # longueur actuelle du serpent (= n. de carrés)
    c = serp[l-1]           # extraction des infos concernant le carré de tête
    xt, yt = c[1], c[2]     # إحداثيات المربع
    # Préparation du déplacement proprement dit.
    # (cc est la taille du carré. dx & dy indiquent le sens du déplacement) :
    xq, yq = xt+dx*cc, yt+dy*cc             # coord. du nouveau carré de tête
    # Vérification : a-t-on atteint les limites du canevas ? :
    if xq<0 or xq>canX-cc or yq<0 or yq>canY-cc:
        flag =0             # => arrêt de l'animation
        can.create_text(canX/2, 20, anchor =CENTER, text ="Perdu !!!",
                        fill ="red", font="Arial 14 bold")
    can.coords(cq, xq, yq, xq+cc, yq+cc)    # déplacement effectif
    serp.append([cq, xq, yq])     # mémorisation du nouveau carré de tête
    del(serp[0])                  # effacement (retrait de la liste)
    # Appel récursif de la fonction par elle-même (=> boucle d'animation) :
    if flag >0:
        fen.after(50, move)

# === البرنامج الرئيسي: ========
```

```
# Variables globales modifiables par certaines fonctions :
flag =0                  # commutateur pour l'animation
dx, dy = 1, 0            # indicateurs pour le sens du déplacement

# Autres variables globales :
canX, canY = 500, 500    # dimensions du canevas
x, y, cc = 100, 100, 15          # coordonnées et coté du premier carré

# Création de l'espace de jeu (fenêtre, canevas, boutons ...) :
fen =Tk()
can =Canvas(fen, bg ='dark gray', height =canX, width =canY)
can.pack(padx =10, pady =10)
bou1 =Button(fen, text="Start", width =10, command =start_it)
bou1.pack(side =LEFT)
bou2 =Button(fen, text="Stop", width =10, command =stop_it)
bou2.pack(side =LEFT)

# Association de gestionnaires d'événements aux touches fléchées du clavier :
fen.bind("<Left>", go_left)         # Attention : les événements clavier
fen.bind("<Right>", go_right)       # doivent toujours être associés à la
fen.bind("<Up>", go_up)             # fenêtre principale, et non au canevas
fen.bind("<Down>", go_down)         # ou à un autre widget.

# Création du serpent initial (= ligne de 5 carrés).
# On mémorisera les infos concernant les carrés créés dans une liste de listes :
serp =[]                            # liste vide
# Création et mémorisation des 5 carrés : le dernier (à droite) est la tête.
i =0
while i <5:
    carre =can.create_rectangle(x, y, x+cc, y+cc, fill="green")
    # Pour chaque carré, on mémorise une petite sous-liste contenant
    # 3 éléments : la référence du carré et ses coordonnées de base :
    serp.append([carre, x, y])
    x =x+cc                  # le carré suivant sera un peu plus à droite
    i =i+1

fen.mainloop()
```

التمرين 9.1 (محرر بسيط، القراءة من ملف والكتابة فيه) :

```
def sansDC(ch):
    "cette fonction renvoie la chaîne ch amputée de son dernier caractère"
    nouv = ""
    i, j = 0, len(ch) -1
    while i < j:
        nouv = nouv + ch[i]
        i = i + 1
    return nouv

def ecrireDansFichier():
    of = open(nomF, 'a')
    while 1:
        ligne = input("entrez une ligne de texte (ou <Enter>) : ")
        if ligne == '':
            break
        else:
            of.write(ligne + '\n')
    of.close()
```

```
def lireDansFichier():
    of = open(nomF, 'r')
    while 1:
        ligne = of.readline()
        if ligne == "":
            break
        # حذف الحرف الأخير (= نهاية الخط):
        print(sansDC(ligne))
    of.close()

nomF = input('Nom du fichier à traiter : ')
choix = input('Entrez "e" pour écrire, "c" pour consulter les données : ')

if choix =='e':
    ecrireDansFichier()
else:
    lireDansFichier()
```

التمرين 9.3 (توليد جداول ضرب 2 × 30) :

```
def tableMulti(n):
    # Fonction générant la table de multiplication par n (20 termes)
    # La table sera renvoyée sous forme d'une chaîne de caractères :
    i, ch = 0, ""
    while i < 20:
        i = i + 1
        ch = ch + str(i * n) + " "
    return ch

NomF = input("Nom du fichier à créer : ")
fichier = open(NomF, 'w')

# Génération des tables de 2 à 30 :
table = 2
while table < 31:
    fichier.write(tableMulti(table) + '\n')
    table = table + 1
fichier.close()
```

التمرين 9.4 :

```
# Triplement des espaces dans un fichier texte.
# Ce script montre également comment modifier le contenu d'un fichier
# en le transférant d'abord tout entier dans une liste, puis en
# ré-enregistrant celle-ci après modifications

def triplerEspaces(ch):
    "fonction qui triple les espaces entre mots dans la chaîne ch"
    i, nouv = 0, ""
    while i < len(ch):
        if ch[i] == " ":
            nouv = nouv + "   "
        else:
            nouv = nouv + ch[i]
        i = i +1
    return nouv

NomF = input("Nom du fichier : ")
```

```
fichier = open(NomF, 'r+')              # 'r+' = وضع القراءة/الكتابة
lignes = fichier.readlines()            # قراءة جميع الأسطر

n=0
while n < len(lignes):
    lignes[n] = triplerEspaces(lignes[n])
    n =n+1

fichier.seek(0)                         # العودة لبداية الملف
fichier.writelines(lignes)              # إعادة
fichier.close()
```

التمرين 9.5 :

```
# Mise en forme de données numériques.
# Le fichier traité est un fichier texte dont chaque ligne contient un nombre
# réel (sans exposants et encodé sous la forme d'une chaîne de caractères)

def valArrondie(ch):
    "représentation arrondie du nombre présenté dans la chaîne ch"
    f = float(ch)       # تحويل السلسلة إلى عدد حقيقي
    e = int(f + .5)     # conversion en entier (On ajoute d'abord
                        # 0.5 au réel pour l'arrondir correctement)
    return str(e)       # التحويل إلى سلسلة

fiSource = input("Nom du fichier à traiter : ")
fiDest = input("Nom du fichier destinataire : ")
fs = open(fiSource, 'r')
fd = open(fiDest, 'w')

while 1:
    ligne = fs.readline()       # قراءة سطر من الملف
    if ligne == "" or ligne == "\n":
        break
    ligne = valArrondie(ligne)
    fd.write(ligne +"\n")

fd.close()
fs.close()
```

التمرين 9.6 :

```
# Comparaison de deux fichiers, caractère par caractère :

fich1 = input("Nom du premier fichier : ")
fich2 = input("Nom du second fichier : ")
fi1 = open(fich1, 'r')
fi2 = open(fich2, 'r')

c, f = 0, 0                 # compteur de caractères et "drapeau"
while 1:
    c = c + 1
    car1 = fi1.read(1)      # lecture d'un caractère dans chacun
    car2 = fi2.read(1)      # des deux fichiers
    if car1 =="" or car2 =="":
        break
    if car1 != car2 :
```

```
        f = 1
        break                # عُثر على الفرق

fi1.close()
fi2.close()

print("Ces 2 fichiers", end=' ')
if f ==1:
    print("diffèrent à partir du caractère n°", c)
else:
    print("sont identiques.")
```

التمرين 9.7 :

```
# مزج ملفين نصيين لملف نصي واحد

fichA = input("Nom du premier fichier : ")
fichB = input("Nom du second fichier : ")
fichC = input("Nom du fichier destinataire : ")
fiA = open(fichA, 'r')
fiB = open(fichB, 'r')
fiC = open(fichC, 'w')

while 1:
    ligneA = fiA.readline()
    ligneB = fiB.readline()
    if ligneA =="" and ligneB =="":
        break                # وصلنا إلى نهاية الملفان
    if ligneA != "":
        fiC.write(ligneA)
    if ligneB != "":
        fiC.write(ligneB)

fiA.close()
fiB.close()
fiC.close()
```

التمرين 9.8 :

```
# تفاصيل محضر أعضاء النادي

def encodage():
    "renvoie la liste des valeurs entrées, ou une liste vide"
    print("*** Veuillez entrer les données (ou <Enter> pour terminer) :")
    while 1:
        nom = input("Nom : ")
        if nom == "":
            return []
        prenom = input("Prénom : ")
        rueNum = input("Adresse (N° et rue) : ")
        cPost = input("Code postal : ")
        local = input("Localité : ")
        tel = input("N° de téléphone : ")
        print(nom, prenom, rueNum, cPost, local, tel)
        ver = input("Entrez <Enter> si c'est correct, sinon <n> ")
        if ver == "":
            break
    return [nom, prenom, rueNum, cPost, local, tel]
```

```
def enregistrer(liste):
    "enregistre les données de la liste en les séparant par des <#>"
    i = 0
    while i < len(liste):
        of.write(liste[i] + "#")
        i = i + 1
    of.write("\n")              # caractère de fin de ligne

nomF = input('Nom du fichier destinataire : ')
of = open(nomF, 'a')
while 1:
    tt = encodage()
    if tt == []:
        break
    enregistrer(tt)

of.close()
```

التمرين 9.9 :

```
# إضافة المعلومات في الملف للنادي

def traduire(ch):
    "convertir une ligne du fichier source en liste de données"
    dn = ""                     # سلسلة مؤقتة لاستخراج البيانات
    tt = []                     # القائمة الناتجة
    i = 0
    while i < len(ch):
        if ch[i] == "#":
            tt.append(dn)       # يتم إضافة البيانات إلى القائمة،
            dn =""              # وإعادة تعيين سلسلة مؤقتة
        else:
            dn = dn + ch[i]
        i = i + 1
    return tt

def encodage(tt):
    "renvoyer la liste tt, complétée avec la date de naissance et le sexe"
    print "*** Veuillez entrer les données (ou <Enter> pour terminer) :"
    # عرض البيانات الموجودة بالفعل في القائمة:
    i = 0
    while i < len(tt):
        print(tt[i], end =' ')
        i = i +1
    print()
    while 1:
        daNai = input("Date de naissance : ")
        sexe = input("Sexe (m ou f) : ")
        print(daNai, sexe)
        ver = input("Entrez <Enter> si c'est correct, sinon <n> ")
        if ver == "":
            break
    tt.append(daNai)
    tt.append(sexe)
    return tt

def enregistrer(tt):
    "enregistrer les données de la liste tt en les séparant par des <#>"
```

```
    i = 0
    while i < len(tt):
        fd.write(tt[i] + "#")
        i = i + 1
    fd.write("\n")          # caractère de fin de ligne

fSource = input('Nom du fichier source : ')
fDest = input('Nom du fichier destinataire : ')
fs = open(fSource, 'r')
fd = open(fDest, 'w')
while 1:
    ligne = fs.readline()           # قراءة سطر من الملف المصدر
    if ligne =="" or ligne =="\n":
        break
    liste = traduire(ligne)         # تحويله إلى قائمة
    liste = encodage(liste)         # y ajouter les données supplémentaires
    enregistrer(liste)              # sauvegarder dans fichier dest.

fd.close()
fs.close()
```

التمرين 9.10 :

```
# Recherche de lignes particulières dans un fichier texte :

def chercheCP(ch):
    "recherche dans ch la portion de chaîne contenant le code postal"
    i, f, ns = 0, 0, 0          # ns est un compteur de codes #
    cc = ""                     # chaîne à construire
    while i < len(ch):
        if ch[i] =="#":
            ns = ns +1
            if ns ==3:          # le CP se trouve après le 3e code #
                f = 1           # variable "drapeau" (flag)
            elif ns ==4:        # inutile de lire après le 4e code #
                break
        elif f ==1:             # le caractère lu fait partie du
            cc = cc + ch[i]     # CP recherché -> on mémorise
        i = i +1
    return cc

nomF = input("Nom du fichier à traiter : ")
codeP = input("Code postal à rechercher : ")
fi = open(nomF, 'r')
while 1:
    ligne = fi.readline()
    if ligne =="":
        break
    if chercheCP(ligne) == codeP:
        print(ligne)
fi.close()
```

التمرين 10.2 (**découpage d'une chaîne en fragments**) :

```
def decoupe(ch, n):
    "découpage de la chaîne ch en une liste de fragments de n caractères"
    d, f = 0, n             # indices de début et de fin de fragment
```

```
    tt = []                  # liste à construire
    while d < len(ch):
        if f > len(ch):      # on ne peut pas découper au-delà de la fin
            f = len(ch)
        fr = ch[d:f]         # découpage d'un fragment
        tt.append(fr)        # ajout du fragment à la liste
        d, f = f, f +n       # indices suivants
    return tt

def inverse(tt):
    "rassemble les éléments de la liste tt dans l'ordre inverse"
    ch = ""                  # chaîne à construire
    i = len(tt)              # on commence par la fin de la liste
    while i > 0 :
        i = i - 1            # le dernier élément possède l'indice n -1
        ch = ch + tt[i]
    return ch

# Test :
if __name__ == '__main__':
    ch ="abcdefghijklmnopqrstuvwxyz123456789âêîôûàèìòùáéíóú"
    liste = decoupe(ch, 5)
    print("chaîne initiale :")
    print(ch)
    print("liste de fragments de 5 caractères :")
    print(liste)
    print("fragments rassemblés après inversion de la liste :")
    print(inverse(liste))
```

التمرينان 10.3 و 10.4 :

```
# Rechercher l'indice d'un caractère donné dans une chaîne

def trouve(ch, car, deb=0):
    "trouve l'indice du caractère car dans la chaîne ch"
    i = deb
    while i < len(ch):
        if ch[i] == car:
            return i    # le caractère est trouvé -> on termine
        i = i + 1
    return -1           # toute la chaîne a été scannée sans succès

# Test :
if __name__ == '__main__':
    print(trouve("Coucou c'est moi", "z"))
    print(trouve("Juliette & Roméo", "&"))
    print(trouve("César & Cléopâtre", "r", 5))
```

التمرين 10.5 :

```
# Comptage des occurrences d'un caractère donné dans une chaîne

def compteCar(ch, car):
    "trouve l'indice du caractère car dans la chaîne ch"
    i, nc = 0, 0              # initialisations
    while i < len(ch):
        if ch[i] == car:
            nc = nc + 1       # caractère est trouvé -> on incrémente le compteur
        i = i + 1
    return nc

# Test :
if __name__ == '__main__':
    print(compteCar("ananas au jus", "a"))
    print(compteCar("Gédéon est déjà là", "é"))
    print(compteCar("Gédéon est déjà là", "à"))
```

التمرين 10.6 :

```
prefixes, suffixe = "JKLMNOP", "ack"

for p in prefixes:
    print(p + suffixe )
```

التمرين 10.7 :

```
def compteMots(ch):
    "comptage du nombre de mots dans la chaîne ch"
    if len(ch) ==0:
        return 0
    nm = 1                   # la chaîne comporte au moins un mot
    for c in ch:
        if c == " ":         # il suffit de compter les espaces
            nm = nm + 1
    return nm
```

```
# Test :
if __name__ == '__main__':
    print(compteMots("Les petits ruisseaux font les grandes rivières"))
```

التمرين 10.8 :

```
def compteCar(ch, car):
    "comptage du nombre de caractères <car> la chaîne <ch>"
    if len(ch) ==0:
        return 0
    n =0
    for c in ch:
        if c == car:
            n = n + 1
    return n

# Programme principal :

def compteCarDeListe(chaine, serie):
    "dans la chaine <ch>, comptage du nombre de caractères listés dans <serie>"
    for cLi in serie:
        nc =compteCar(chaine, cLi)
        print("Caractère", cLi, ":", nc)


# Test :
if __name__ == '__main__':
    txt ="René et Célimène étaient eux-mêmes nés à Noël de l'année dernière"
    print(txt)
    compteCarDeListe(txt, "eéèêë")
```

التمرين 10.9 :

```
def estUnChiffre(car):
    "renvoie <vrai> si le caractère 'car' est un chiffre"
    if car in "0123456789":
        return "vrai"
    else:
        return "faux"

# Test :
if __name__ == '__main__':
    caracteres ="d75è8b0â1"
    print("Caractères à tester :", caracteres)
    for car in caracteres:
        print(car, estUnChiffre(car))
```

التمرين 10.10 :

```
def estUneMaj(car):
    "renvoie <vrai> si le caractère 'car' est une majuscule"
    if car in "ABCDEFGHIJKLMNOPQRSTUVWXYZÀÂÉÈÊËÇÎÏÙÜÛÔÖ":
        return True
    else:
        return False
```

```
# Test :
if __name__ == '__main__':
    caracteres ="eÀçMöSÖÛmÇéùT"
    print("Caractères à tester :", caracteres)
    for car in caracteres:
        print(car, estUneMaj(car))
```

التمرين 10.11 :

```
def chaineListe(ch):
    "convertit la chaîne ch en une liste de mots"
    liste, ct = [], ""          # ct est une chaîne temporaire
    for c in ch:                # examiner tous les caractères de ch
        if c == " ":            # lorsqu'on rencontre un espace,
            liste.append(ct)    # on ajoute la chaîne temporaire à la liste
            ct = ""             # ... et on ré-initialise la chaîne temporaire
        else:
            # les autres caractères examinés sont ajoutés à la chaîne temp. :
            ct = ct + c
    # Ne pas oublier le mot restant après le dernier espace ! :
    if ct:                      # vérifier si ct n'est pas une chaîne vide
        liste.append(ct)
    return liste                # renvoyer la liste ainsi construite

# Tests :
if __name__ == '__main__':
    li = chaineListe("René est un garçon au caractère héroïque")
    print(li)
    for mot in li:
        print(mot, "-", end=' ')
    print(chaineListe(""))              # doit renvoyer une liste vide
```

التمرين 10.12 (**utilise les deux fonctions définies dans les exercices précédents**) :

```
from exercice_10_10 import estUneMaj
from exercice_10_11 import chaineListe

txt = "Le prénom de cette Dame est Élise"
print("Phrase à tester :", txt)

lst = chaineListe(txt)          # convertir la phrase en une liste de mots

for mot in lst:                 # analyser chacun des mots de la liste
    prem = mot[0]               # extraction du premier caractère
    if estUneMaj(prem):         # test de majuscule
        print(mot)

# Variante plus compacte, utilisant la composition :
print("Variante :")
for mot in lst:
    if estUneMaj(mot[0]):
        print(mot)
```

التمرين 10.13 (**utilise les deux fonctions définies dans les exercices précédents**) :

```
from exercice_10_10 import estUneMaj
```

```
from exercice_10_11 import chaineListe

def compteMaj(ch):
    "comptage des mots débutant par une majuscule dans la chaîne ch"
    c = 0
    lst = chaineListe(ch)       # convertir la phrase en une liste de mots
    for mot in lst:             # analyser chacun des mots de la liste
        if estUneMaj(mot[0]):
            c = c +1
    return c

# Test :
if __name__ == '__main__':
    phrase = "Les filles Tidgoutt se nomment Joséphine, Justine et Corinne"
    print("Phrase à tester : ", phrase)
    print("Cette phrase contient", compteMaj(phrase), "majuscules.")
```

التمرين 10.14 (جدول محارف **ASCII**) :

```
# Table des codes ASCII

c = 32              # premier code ASCII <imprimable>

while c < 128 :     # dernier code strictement ASCII = 127
    print("Code", c, ":", chr(c), end =" - ")
    c = c + 1
```

التمرين 10.16 (**échange des majuscules et des minuscules**) :

```
def convMajMin(ch):
    "échange les majuscules et les minuscules dans la chaîne ch"
    nouvC = ""                              # chaîne à construire
    for car in ch:
        code = ord(car)
        # les codes numériques des caractères majuscules et minuscules
        # correspondants sont séparés de 32 unités :
        if code >= 65 and code <= 91:       # majuscules ordinaires
            code = code + 32
        elif code >= 192 and code <= 222:   # majuscules accentuées
            code = code + 32
        elif code >= 97 and code <= 122:    # minuscules ordinaires
            code = code - 32
        elif code >= 224 and code <= 254:   # minuscules accentuées
            code = code - 32
        nouvC = nouvC + chr(code)
    # renvoi de la chaîne construite :
    return nouvC

# test :
if __name__ == '__main__':
    txt ="Émile Noël épouse Irène Müller"
    print(txt)
    print(convMajMin(txt))
```

التمرين 10.17 (تحويل **Latin-1** إلى **Utf-8**) :

```
# Traitement et conversion de lignes dans un fichier texte

def traiteLigne(ligne):
    "remplacement des espaces de la ligne de texte par '-*-' "
    newLine =""                          # nouvelle chaîne à construire
    c, m = 0, 0                          # initialisations
    while c < len(ligne):                # lire tous les caractères de la ligne
        if ligne[c] == " ":
            # Le caractère lu est un espace.
            # On ajoute une 'tranche' à la chaîne en cours de construction :
            newLine = newLine + ligne[m:c] + "-*-"
            # On mémorise dans m la position atteinte dans la ligne lue :
            m = c + 1                    # ajouter 1 pour "oublier" l'espace
        c = c + 1
    # Ne pas oublier d'ajouter la 'tranche' suivant le dernier espace :
    newLine = newLine + ligne[m:]
    # Renvoyer la chaîne construite :
    return newLine

# --- Programme principal : ---
nomFS = input("Nom du fichier source (Latin-1) : ")
nomFD = input("Nom du fichier destinataire (Utf-8) : ")
fs = open(nomFS, 'r', encoding ="Latin1")     # ouverture des 2 fichiers
fd = open(nomFD, 'w', encoding ="Utf8")       # dans les encodages spécifiés
while 1:                                    # boucle de traitement
    li = fs.readline()                      # lecture d'une ligne
    if li == "":                            # détection de la fin du fichier :
        break                               # readline() renvoie une chaîne vide
    fd.write(traiteLigne(li))               # traitement + écriture
fd.close()
fs.close()
```

التمرين 10.18 (**tester si un caractère donné est une voyelle**) :

```
def voyelle(car):
    "teste si le caractère <car> est une voyelle"
    if car in "AEIOUYÀÉÈÊËÎÏÔÛÙaeiouyàéèêëîïôûù":
        return True
    else:
        return False

# Test :
if __name__ == '__main__':
    ch ="gOàÉsùïÇ"                  # lettres à tester
    for c in ch:
        print(c, ":", voyelle(c))
```

التمرين 10.19 (**utilise la fonction définie dans le script précédent**) :

```
from exercice_10_18 import voyelle

def compteVoyelles(phrase):
    "compte les voyelles présentes dans la chaîne de caractères <phrase>"
    n = 0
```

```
    for c in phrase:
        if voyelle(c):
            n = n + 1
    return n

# Test :
if __name__ == '__main__':
    texte ="Maître corbeau sur un arbre perché"
    nv = compteVoyelles(texte)
    print("La phrase <", texte, "> compte ", nv, " voyelles.", sep="")
```

التمرين 10.20 :

```
c = 1040                        # code du premier caractère (majuscule)
maju =""                        # chaîne destinée aux majuscules
minu =""                        # chaîne destinée aux minuscules
while c <1072:                  # on se limitera à cette gamme
    maju = maju + chr(c)
    minu = minu + chr(c +32)    # voir exercices précédents
    c = c+1
print(maju)
print(minu)
```

التمرين 10.21 :

```
# Conversion en majuscule du premier caractère de chaque mot dans un texte.

fiSource = input("Nom du fichier à traiter (Latin-1) : ")
fiDest = input("Nom du fichier destinataire (Utf-8) : ")
fs = open(fiSource, 'r', encoding ="Latin1")
fd = open(fiDest, 'w', encoding ="Utf8")

while 1:
    ch = fs.readline()                  # lecture d'une ligne
    if ch == "":
        break                           # fin du fichier
    ch = ch.title()                     # conversion des initiales en maj.
    fd.write(ch)                        # transcription

fd.close()
fs.close()
```

التمرين 10.22 :

```
# Conversion Latin-1 => Utf8 (variante utilisant une variable <bytes>

fiSource = input("Nom du fichier à traiter (Latin-1) : ")
fiDest = input("Nom du fichier destinataire (Utf-8) : ")
fs = open(fiSource, 'rb')               # mode de lecture <binaire>
fd = open(fiDest, 'wb')                 # mode d'écriture <binaire>

while 1:
    so = fs.readline()                  # la ligne lue est une séquence d'octets
    # Remarque : la variable so étant du type <bytes>, on doit la comparer
    # avec une chaîne littérale (vide) du même type dans les tests :
    if so == b"":
```

```
        break                       # fin du fichier
    ch = so.decode("Latin-1")       # conversion en chaîne de caractères
    ch = ch.replace(" ","-*-")      # remplacement des espaces par -*-
    so = ch.encode("Utf-8")         # Ré-encodage en une séquence d'octets
    fd.write(so)                    # transcription

fd.close()
fs.close()
```

التمرين 10.23 :

```
# Comptage du nombre de mots dans un texte

fiSource = input("Nom du fichier à traiter : ")
fs = open(fiSource, 'r')

n = 0                          # variable compteur
while 1:
    ch = fs.readline()
    if ch == "":               # fin du fichier
        break
    # conversion de la chaîne lue en une liste de mots :
    li = ch.split()
    # totalisation des mots :
    n = n + len(li)
fs.close()

print("Ce fichier texte contient un total de %s mots" % (n))
```

التمرين 10.24 :

```
# Fusion de lignes pour former des phrases

fiSource = input("Nom du fichier à traiter (Latin-1) : ")
fiDest = input("Nom du fichier destinataire (Utf-8) : ")
fs = open(fiSource, 'r', encoding ="Latin1")
fd = open(fiDest, 'w', encoding ="Utf8")

# On lit d'abord la première ligne :
ch1 = fs.readline()
# On lit ensuite les suivantes, en les fusionnant si nécessaire :
while 1:
    ch2 = fs.readline()
    if not ch2:           # Rappel : une chaîne vide est considérée
        break             # comme "fausse" dans les tests
    # Si la chaîne lue commence par une majuscule, on transcrit
    # la précédente dans le fichier destinataire, et on la
    # remplace par celle que l'on vient de lire :
    if ch2[0] in "ABCDEFGHIJKLMNOPQRSTUVWXYZÀÂÉÈÊËÎÏÔÙÛÇ":
        fd.write(ch1)
        ch1 = ch2
    # Sinon, on la fusionne avec la précédente, en veillant à en
    # enlever au préalable le ou les caractère(s) de fin de ligne.
    else:
        ch1 = ch1[:-1] + " " + ch2

# Attention : ne pas oublier de transcrire la dernière ligne :
fd.write(ch1)
```

```
fd.close()
fs.close()
```

التمرين 10.25 (**caractéristiques de sphères**) :

```
# Le fichier de départ est un fichier <texte> dont chaque ligne contient
# un nombre réel (encodé sous la forme d'une chaîne de caractères)

from math import pi

def caractSphere(d):
    "renvoie les caractéristiques d'une sphère de diamètre d"
    d = float(d)        # conversion de l'argument (=chaîne) en réel
    r = d/2             # rayon
    ss = pi*r**2        # surface de section
    se = 4*pi*r**2      # surface extérieure
    v = 4/3*pi*r**3     # volume
    # La balise {:8.2f} utilisé ci-dessous formate le nombre
    # affiché de manière à occuper 8 caractères au total, en arrondissant
    # de manière à conserver deux chiffres après la virgule :
    ch = "Diam. {:6.2f} cm Section = {:8.2f} cm² ".format(d, ss)
    ch = ch +"Surf. = {:8.2f} cm². Vol. = {:9.2f} cm³".format(se, v)
    return ch

fiSource = input("Nom du fichier à traiter : ")
fiDest = input("Nom du fichier destinataire : ")
fs = open(fiSource, 'r')
fd = open(fiDest, 'w')
while 1:
    diam = fs.readline()
    if diam == "" or diam == "\n":
        break
    fd.write(caractSphere(diam) + "\n")        # تدوين
fd.close()
fs.close()
```

التمرين 10.26 :

```
# Mise en forme de données numériques
# Le fichier traité est un fichier <texte> dont chaque ligne contient un nombre
# réel (sans exposants et encodé sous la forme d'une chaîne de caractères)

def arrondir(reel):
    "représentation arrondie à .0 ou .5 d'un nombre réel"
    ent = int(reel)             # partie entière du nombre
    fra = reel - ent            # partie fractionnaire
    if fra < .25 :
        fra = 0
    elif fra < .75 :
        fra = .5
    else:
        fra = 1
    return ent + fra

fiSource = input("Nom du fichier à traiter : ")
fiDest = input("Nom du fichier destinataire : ")
fs = open(fiSource, 'r')
fd = open(fiDest, 'w')
```

```
while 1:
    ligne = fs.readline()
    if ligne == "" or ligne == "\n":
        break
    n = arrondir(float(ligne))      # conversion en <float>, puis arrondi
    fd.write(str(n) + "\n")         # تدوين

fd.close()
fs.close()
```

التمرين 10.29 :

```
# Affichage de tables de multiplication

nt = [2, 3, 5, 7, 9, 11, 13, 17, 19]

def tableMulti(m, n):
     "renvoie n termes de la table de multiplication par m"
     ch =""
     for i in range(n):
          v = m * (i+1)               # calcul d'un des termes
          ch = ch + "%4d" % (v)       # formatage à 4 caractères
     return ch

for a in nt:
     print(tableMulti(a, 15))         # 15 premiers termes seulement
```

التمرين 10.30 (**simple parcours d'une liste**) :

```
# -*- coding:Utf-8 -*-

lst = ['Jean-Michel', 'Marc', 'Vanessa', 'Anne',
       'Maximilien', 'Alexandre-Benoît', 'Louise']

for e in lst:
     print("%s : %s caractères" % (e, len(e)))
```

التمرين 10.31 :

```
# Élimination de doublons

lst = [9, 12, 40, 5, 12, 3, 27, 5, 9, 3, 8, 22, 40, 3, 2, 4, 6, 25]
lst2 = []

for el in lst:
     if el not in lst2:
          lst2.append(el)
lst2.sort()

print("Liste initiale :", lst)
print("Liste traitée  :", lst2)
```

التمرين 10.33 (عرض كل أيام السنة) :

```
## Cette variante utilise une liste de listes ##
## (que l'on pourrait aisément remplacer par deux listes distinctes)

# La liste ci-dessous contient deux éléments qui sont eux-mêmes des listes.
# l'élément 0 contient les nombres de jours de chaque mois, tandis que
# l'élément 1 contient les noms des douze mois :
mois = [[31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31],
        ['Janvier', 'Février', 'Mars', 'Avril', 'Mai', 'Juin', 'Juillet',
         'Août', 'Septembre', 'Octobre', 'Novembre', 'Décembre']]

jour = ['Dimanche','Lundi','Mardi','Mercredi','Jeudi','Vendredi','Samedi']

ja, jm, js, m = 0, 0, 0, 0

while ja <365:
    ja, jm = ja +1, jm +1     # ja = jour dans l'année, jm = jour dans le mois
    js = (ja +3) % 7          # js = jour de la semaine. Le décalage ajouté
                              #      permet de choisir le jour de départ

    if jm > mois[0][m]:                # élément m de l'élément 0 de la liste
        jm, m = 1, m+1

    print(jour[js], jm, mois[1][m])   # élément m de l'élément 1 de la liste
```

التمرين 10.36 :

```
# Insertion de nouveaux éléments dans une liste existante

t1 = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
t2 = ['Janvier','Février','Mars','Avril','Mai','Juin',
      'Juillet','Août','Septembre','Octobre','Novembre','Décembre']

c, d = 1, 0
while d < 12 :
     t2[c:c] = [t1[d]]        # ! l'élément inséré doit être une liste
     c, d = c+2, d+1

print(t2)
```

التمرين 10.40 :

```
# Crible d'Eratosthène pour rechercher les nombres premiers de 1 à 999

# Créer une liste de 1000 éléments 1 (leurs indices vont de 0 à 999) :
lst = [1]*1000
# Parcourir la liste à partir de l'élément d'indice 2:
for i in range(2,1000):
    # Mettre à zéro les éléments suivants dans la liste,
    # dont les indices sont des multiples de i :
    for j in range(i*2, 1000, i):
        lst[j] = 0

# Afficher les indices des éléments restés à 1 (on ignore l'élément 0) :
for i in range(1,1000):
    if lst[i]:
        print(i, end =’ ‘)
```

التمرين 10.43 (اختبار مولد أرقام عشوائية) :

```
from random import random          # tire au hasard un réel entre 0 et 1

n = input("Nombre de valeurs à tirer au hasard (défaut = 1000) : ")
if n == "":
    nVal =1000
else:
    nVal = int(n)
n = input("Nombre de fractions dans l'intervalle 0-1 (entre 2 et {0}, "\
          "défaut =5) : ".format(nVal//10))
if n == "":
    nFra =5
else:
    nFra = int(n)
if nFra < 2:
    nFra =2
elif nFra > nVal/10:
    nFra = nVal/10

print("Tirage au sort des", nVal, "valeurs ...")
listVal = [0]*nVal                      # créer une liste de zéros
for i in range(nVal):                   # puis modifier chaque élément
    listVal[i] = random()
print("Comptage des valeurs dans chacune des", nFra, "fractions ...")
listCompt = [0]*nFra                    # créer une liste de compteurs

# parcourir la liste des valeurs :
for valeur in listVal:
    # trouver l'index de la fraction qui contient la valeur :
    index = int(valeur*nFra)
    # incrémenter le compteur correspondant :
    listCompt[index] = listCompt[index] +1

# afficher l'état des compteurs :
for compt in listCompt:
    print(compt, end =' ')
print()
```

التمرين 10.44 : رسم بطاقات

```
from random import randrange

couleurs = ['Pique', 'Trèfle', 'Carreau', 'Cœur']
valeurs = [2, 3, 4, 5, 6, 7, 8, 9, 10, 'valet', 'dame', 'roi', 'as']

# Construction de la liste des 52 cartes :
carte =[]
for coul in couleurs:
     for val in valeurs:
          carte.append("{0} de {1}".format(val, coul))

# Tirage au hasard :
while 1:
     k = input("Frappez <c> pour tirer une carte, <Enter> pour terminer ")
     if k =="":
          break
     r = randrange(52)      # tirage au hasard d'un entier entre 0 et 51
```

```
    print(carte[r])
```

التمرين 10.45 : Création et consultation d'un dictionnaire

```
# Mini système de bases de données

def consultation():
    while 1:
        nom = input("Entrez le nom (ou <enter> pour terminer) : ")
        if nom == "":
            break
        if nom in dico:                  # le nom est-il répertorié ?
            item = dico[nom]             # consultation proprement dite
            age, taille = item[0], item[1]
            print("Nom : {0} - âge : {1} ans - taille : {2} m.".\
                  format(nom, age, taille))
        else:
            print("*** nom inconnu ! ***")

def remplissage():
    while 1:
        nom = input("Entrez le nom (ou <enter> pour terminer) : ")
        if nom == "":
            break
        age = int(input("Entrez l'âge (nombre entier !) : "))
        taille = float(input("Entrez la taille (en mètres) : "))
        dico[nom] = (age, taille)

dico ={}
while 1:
    choix = input("Choisissez : (R)emplir - (C)onsulter - (T)erminer : ")
    if choix.upper() == 'T':
        break
    elif choix.upper() == 'R':
        remplissage()
    elif choix.upper() == 'C':
        consultation()
```

التمرين 10.46 : échange des clés et des valeurs dans un dictionnaire

```
def inverse(dico):
    "Construction d'un nouveau dico, pas à pas"
    dic_inv ={}
    for cle in dico:
        item = dico[cle]
        dic_inv[item] = cle

    return dic_inv

# programme test :

dico = {'Computer':'Ordinateur',
        'Mouse':'Souris',
        'Keyboard':'Clavier',
        'Hard disk':'Disque dur',
        'Screen':'Écran'}
```

```
print(dico)
print(inverse(dico))
```

التمرين 10.47 : رسم بياني

```
# Histogramme des fréquences de chaque lettre dans un texte

nFich = input('Nom du fichier (Latin-1) : ')
fi = open(nFich, 'r', encoding ="Latin1")
texte = fi.read()
fi.close()

print(texte)
dico ={}
for c in texte:                        # afin de les regrouper, on convertit
    c = c.upper()                       # toutes les lettres en majuscules
    dico[c] = dico.get(c, 0) +1

liste = list(dico.items())
liste.sort()
for car, freq in liste:
    print("Caractère {0} : {1} occurrence(s).".format(car, freq))
```

التمرين 10.48 :

```
# Histogramme des fréquences de chaque mot dans un texte
# Suivant l'encodage du fichier source, activer l'une ou l'autre ligne :
encodage ="Latin-1"
# encodage ="Utf-8"

nFich = input('Nom du fichier à traiter ({0}) : '.format(encodage))
# Conversion du fichier en une chaîne de caractères :
fi = open(nFich, 'r', encoding =encodage)
texte = fi.read()
fi.close()

# afin de pouvoir aisément séparer les mots du texte, on commence
# par convertir tous les caractères non-alphabétiques en espaces  :
alpha = "abcdefghijklmnopqrstuvwxyzéèàùçâêîôûäëïöü"
lettres = ""            # nouvelle chaîne à construire
for c in texte:
    c = c.lower()       # conversion de chaque caractère en minuscule
    if c in alpha:
        lettres = lettres + c
    else:
        lettres = lettres + ' '

# conversion de la chaîne résultante en une liste de mots :
mots = lettres.split()

# construction de l'histogramme :
dico ={}
for m in mots:
    dico[m] = dico.get(m, 0) +1
liste = list(dico.items())

# tri de la liste résultante :
liste.sort()
```

```
# affichage en clair :
for item in liste:
    print("{0} : {1}".format(item[0], item[1]))
```

التمرين 10.49 :

```
# Encodage d'un texte dans un dictionnaire
# Suivant l'encodage du fichier source, activer l'une ou l'autre ligne :
encodage ="Latin-1"
# encodage ="Utf-8"

nFich = input('Nom du fichier à traiter ({0}) : '.format(encodage))
# Conversion du fichier en une chaîne de caractères :
fi = open(nFich, 'r', encoding =encodage)
texte = fi.read()
fi.close()

# On considère que les mots sont des suites de caractères faisant partie
# de la chaîne ci-dessous. Tous les autres sont des séparateurs :
alpha = "abcdefghijklmnopqrstuvwxyzéèàùçâêîôûäëïöü"

# Construction du dictionnaire :
dico ={}
# Parcours de tous les caractères du texte :
i =0                         # indice du caractère en cours de lecture
im =-1                       # indice du premier caractère du mot
mot = ""                     # variable de travail : mot en cours de lecture
for c in texte:
    c = c.lower()            # conversion de chaque caractère en minuscule

    if c in alpha:           # car. alphabétique => on est à l'intérieur d'un mot
        mot = mot + c
        if im < 0:           # mémoriser l'indice du premier caractère du mot
            im =i
    else:                    # car. non-alphabétique => fin de mot
        if mot != "":        # afin d'ignorer les car. non-alphab. successifs
            # pour chaque mot, on construit une liste d'indices :
            if mot in dico:               # mot déjà répertorié :
                dico[mot].append(im)      # ajout d'un indice à la liste
            else:                         # mot rencontré pour la 1e fois :
                dico[mot] =[im]           # création de la liste d'indices
            mot =""          # préparer la lecture du mot suivant
            im =-1
    i += 1                   # indice du caractère suivant

# Affichage du dictionnaire, en clair :
listeMots =list(dico.items())      # Conversion du dico en une liste de tuples
listeMots.sort()                   # tri alphabétique de la liste
for clef, valeur in listeMots:
    print(clef, ":", valeur)
```

التمرين 10.50 : **Sauvegarde d’un dictionnaire (complément de l’ex. 10.45)**.

```
# Mini-système de base de données

def consultation():
    while 1:
        nom = input("Entrez le nom (ou <enter> pour terminer) : ")
```

```
        if nom == "":
            break
        if nom in dico:                 # le nom est-il répertorié ?
            item = dico[nom]            # consultation proprement dite
            age, taille = item[0], item[1]
            print("Nom : {0} - âge : {1} ans - taille : {2} m.".\
                  format(nom, age, taille))
        else:
            print("*** nom inconnu ! ***")

def remplissage():
    while 1:
        nom = input("Entrez le nom (ou <enter> pour terminer) : ")
        if nom == "":
            break
        age = int(input("Entrez l'âge (nombre entier !) : "))
        taille = float(input("Entrez la taille (en mètres) : "))
        dico[nom] = (age, taille)

def enregistrement():
    fich = input("Entrez le nom du fichier de sauvegarde : ")
    ofi = open(fich, "w")
    # écriture d'une ligne-repère pour identifier le type de fichier :
    ofi.write("DicoExercice10.50\n")
    # parcours du dictionnaire entier, converti au préalable en une liste :
    for cle, valeur in list(dico.items()):
        # utilisation du formatage des chaînes pour créer l'enregistrement :
        ofi.write("{0}@{1}#{2}\n".format(cle, valeur[0], valeur[1]))
    ofi.close()

def lectureFichier():
    fich = input("Entrez le nom du fichier de sauvegarde : ")
    try:
        ofi = open(fich, "r")
    except:
        print("*** fichier inexistant ***")
        return
    # Vérification : le fichier est-il bien de notre type spécifique ? :
    repere =ofi.readline()
    if repere != "DicoExercice10.50\n":
        print("*** type de fichier incorrect ***")
        return
    # Lecture des lignes restantes du fichier :
    while 1:
        ligne = ofi.readline()
        if ligne =='':              # détection de la fin de fichier
            break
        enreg = ligne.split("@")    # restitution d'une liste [clé,valeur]
        cle = enreg[0]
        valeur = enreg[1][:-1]      # élimination du caractère de fin de ligne
        data = valeur.split("#")    # restitution d'une liste [âge, taille]
        age, taille = int(data[0]), float(data[1])
        dico[cle] = (age, taille)   # reconstitution du dictionnaire
    ofi.close()

########### Programme principal : ###########
dico ={}
lectureFichier()
while 1:
    choix = input("Choisissez : (R)emplir - (C)onsulter - (T)erminer : ")
    if choix.upper() == 'T':
```

```
        break
    elif choix.upper() == 'R':
        remplissage()
    elif choix.upper() == 'C':
        consultation()
enregistrement()
```

التمرين 10.51 : Contrôle du flux d'exécution à l'aide d'un dictionnaire

Cet exercice complète le précédent. On ajoute encore deux petites fonctions, et on réécrit le corps principal du programme pour diriger le flux d'exécution en se servant d'un dictionnaire :

```
def sortie():
    print("*** Job terminé ***")
    return 1                        # afin de provoquer la sortie de la boucle

def autre():
    print("Veuillez frapper R, A, C, S ou T, svp.")

######## * Programme principal * #########

dico ={}
fonc ={"R":lectureFichier, "A":remplissage, "C":consultation,
       "S":enregistrement, "T":sortie}
while 1:
    choix = input("Choisissez :\n" +\
    "(R)écupérer un dictionnaire préexistant sauvegardé dans un fichier\n" +\
    "(A)jouter des données au dictionnaire courant\n" +\
    "(C)onsulter le dictionnaire courant\n" +\
    "(S)auvegarder le dictionnaire courant dans un fichier\n" +\
    "(T)erminer : ").upper()
    # l'instruction ci-dessous appelle une fonction différente pour chaque
    # choix, par l'intermédiaire du dictionnaire <fonc> :
    if fonc.get(choix, autre)():
        break
    # note : toutes les fonctions appelées ici renvoient <None> par défaut
    #         sauf la fonction sortie() qui renvoie 1 => sortie de la boucle
```

التمرين 11.1 :

```
from math import sqrt               # fonction racine carrée

def distance(p1, p2):
    # On applique le théorème de Pythagore :
    dx =abs(p1.x - p2.x)            # abs() => valeur absolue
    dy =abs(p1.y - p2.y)
    return sqrt(dx*dx + dy*dy)

def affiche_point(p):
    print("Coord. horiz.", p.x, "Coord. vert.", p.y)

class Point(object):
    "Classe de points géométriques"

# Définition des 2 points :
p8, p9 = Point(), Point()
p8.x, p8.y, p9.x, p9.y = 12.3, 5.7, 6.2, 9.1
```

```
affiche_point(p8)
affiche_point(p9)
print("Distance =", distance(p8,p9))
```

التمرين 12.1 :

```
class Domino(object):
    def __init__(self, pa, pb):
        self.pa, self.pb = pa, pb

    def affiche_points(self):
        print "face A :", self.pa,
        print "face B :", self.pb

    def valeur(self):
        return self.pa + self.pb

# Programme de test :

d1 = Domino(2,6)
d2 = Domino(4,3)

d1.affiche_points()
d2.affiche_points()

print("total des points :", d1.valeur() + d2.valeur())

liste_dominos = []
for i in range(7):
    liste_dominos.append(Domino(6, i))

vt =0
for i in range(7):
    liste_dominos[i].affiche_points()
    vt = vt + liste_dominos[i].valeur()

print("valeur totale des points", vt)
print(liste_dominos[3], liste_dominos[4])
```

التمرين 12.2 :

```
class CompteBancaire(object):
    def __init__(self, nom ='Dupont', solde =1000):
        self.nom, self.solde = nom, solde

    def depot(self, somme):
        self.solde = self.solde + somme

    def retrait(self, somme):
        self.solde = self.solde - somme

    def affiche(self):
        print("Le solde du compte bancaire de {0} est de {1} euros.".\
              format(self.nom, self.solde))

# Programme de test :

if __name__ == '__main__':
```

```
    c1 = CompteBancaire('Duchmol', 800)
    c1.depot(350)
    c1.retrait(200)
    c1.affiche()
```

التمرين 12.3 :

```
class Voiture(object):
    def __init__(self, marque = 'Ford', couleur = 'rouge'):
        self.couleur = couleur
        self.marque = marque
        self.pilote = 'personne'
        self.vitesse = 0

    def accelerer(self, taux, duree):
        if self.pilote =='personne':
            print("Cette voiture n'a pas de conducteur !")
        else:
            self.vitesse = self.vitesse + taux * duree

    def choix_conducteur(self, nom):
        self.pilote = nom

    def affiche_tout(self):
        print("{0} {1} pilotée par {2}, vitesse = {3} m/s".\
            format(self.marque, self.couleur, self.pilote, self.vitesse))

a1 = Voiture('Peugeot', 'bleue')
a2 = Voiture(couleur = 'verte')
a3 = Voiture('Mercedes')
a1.choix_conducteur('Roméo')
a2.choix_conducteur('Juliette')
a2.accelerer(1.8, 12)
a3.accelerer(1.9, 11)
a2.affiche_tout()
a3.affiche_tout()
```

التمرين 12.4 :

```
class Satellite(object):
    def __init__(self, nom, masse =100, vitesse =0):
        self.nom, self.masse, self.vitesse = nom, masse, vitesse

    def impulsion(self, force, duree):
        self.vitesse = self.vitesse + force * duree / self.masse

    def energie(self):
        return self.masse * self.vitesse**2 / 2

    def affiche_vitesse(self):
        print("Vitesse du satellite {0} = {1} m/s".\
              format(self.nom, self.vitesse))

# Programme de test :

s1 = Satellite('Zoé', masse =250, vitesse =10)

s1.impulsion(500, 15)
```

```
s1.affiche_vitesse()
print("énergie =", s1.energie())
s1.impulsion(500, 15)
s1.affiche_vitesse()
print("nouvelle énergie =", s1.energie())
```

التمرينان 12.5-12.6 (أصناف الاسطوانات والمخاريط) :

```
# Classes dérivées - Polymorphisme

class Cercle(object):
    def __init__(self, rayon):
        self.rayon = rayon

    def surface(self):
        return 3.1416 * self.rayon**2

class Cylindre(Cercle):
    def __init__(self, rayon, hauteur):
        Cercle.__init__(self, rayon)
        self.hauteur = hauteur

    def volume(self):
        return self.surface()*self.hauteur

    # la méthode surface() est héritée de la classe parente

class Cone(Cylindre):
    def __init__(self, rayon, hauteur):
        Cylindre.__init__(self, rayon, hauteur)

    def volume(self):
        return Cylindre.volume(self)/3
        # cette nouvelle méthode volume() remplace celle que
        # l'on a héritée de la classe parente (exemple de polymorphisme)

# Programme test :

cyl = Cylindre(5, 7)
print("Surf. de section du cylindre =", cyl.surface())
print("Volume du cylindre =", cyl.volume())

co = Cone(5,7)
print("Surf. de base du cône =", co.surface())
print("Volume du cône =", co.volume())
```

التمرين 12.7 :

```
# Tirage de cartes

from random import randrange

class JeuDeCartes(object):
    """Jeu de cartes"""
    # attributs de classe (communs à toutes les instances) :
    couleur = ('Pique', 'Trèfle', 'Carreau', 'Cœur')
    valeur = (0, 0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 'valet', 'dame', 'roi', 'as')
```

```
    def __init__(self):
        "Construction de la liste des 52 cartes"
        self.carte =[]
        for coul in range(4):
            for val in range(13):
                self.carte.append((val +2, coul))    # la valeur commence à 2

    def nom_carte(self, c):
        "Renvoi du nom de la carte c, en clair"
        return "{0} de {1}".format(self.valeur[c[0]], self.couleur[c[1]])

    def battre(self):
        "Mélange des cartes"
        t = len(self.carte)             # nombre de cartes restantes
        # pour mélanger, on procède à un nombre d'échanges équivalent :
        for i in range(t):
            # tirage au hasard de 2 emplacements dans la liste :
            h1, h2 = randrange(t), randrange(t)
            # échange des cartes situées à ces emplacements :
            self.carte[h1], self.carte[h2] = self.carte[h2], self.carte[h1]

    def tirer(self):
        "Tirage de la première carte de la pile"
        t = len(self.carte)             # vérifier qu'il reste des cartes
        if t >0:
            carte = self.carte[0]       # choisir la première carte du jeu
            del(self.carte[0])          # la retirer du jeu
            return carte                # en renvoyer copie au prog. appelant
        else:
            return None                 # facultatif

### Programme test :

if __name__ == '__main__':
    jeu = JeuDeCartes()                 # instanciation d'un objet
    jeu.battre()                        # mélange des cartes
    for n in range(53):                 # tirage des 52 cartes :
        c = jeu.tirer()
        if c == None:                   # il ne reste aucune carte
            print('Terminé !')          # dans la liste
        else:
            print(jeu.nom_carte(c))     # valeur et couleur de la carte
```

التمرين 12.8 :

On supposera que l'exercice précédent a été sauvegardé sous le nom (***cartes.py***.)

```
# Bataille de de cartes

from cartes import JeuDeCartes

jeuA = JeuDeCartes()        # instanciation du premier jeu
jeuB = JeuDeCartes()        # instanciation du second jeu
jeuA.battre()               # mélange de chacun
jeuB.battre()
pA, pB = 0, 0               # compteurs de points des joueurs A et B
```

```
# tirer 52 fois une carte de chaque jeu :
for n in range(52):
    cA, cB = jeuA.tirer(), jeuB.tirer()
    vA, vB = cA[0], cB[0]   # valeurs de ces cartes
    if vA > vB:
        pA += 1
    elif vB > vA:
        pB += 1             # (rien ne se passe si vA = vB)
    # affichage des points successifs et des cartes tirées :
    print("{0} * {1} ==> {2} * {3}".format(jeuA.nom_carte(cA),
                                       jeuB.nom_carte(cB), pA, pB))

print("le joueur A obtient {0} pts, le joueur B en obtient {1}.".format(pA, pB))
```

التمرين 12.9 :

```
from exercice_12_02 import CompteBancaire

class CompteEpargne(CompteBancaire):
    def __init__(self, nom ='Durand', solde =500):
        CompteBancaire.__init__(self, nom, solde)
        self.taux =.3          # taux d'intérêt mensuel par défaut

    def changeTaux(self, taux):
        self.taux =taux

    def capitalisation(self, nombreMois =6):
        print("Capitalisation sur {0} mois au taux mensuel de {1} %.".\
              format(nombreMois, self.taux))
        for m in range(nombreMois):
            self.solde = self.solde * (100 +self.taux)/100

# Programme de test :

if __name__ == '__main__':
    c1 = CompteEpargne('Duvivier', 600)
    c1.depot(350)
    c1.affiche()
    c1.capitalisation(12)
    c1.affiche()
    c1.changeTaux(.5)
    c1.capitalisation(12)
    c1.affiche()
```

التمرين 13.6 :

```
from tkinter import *

def cercle(can, x, y, r, coul ='white'):
    "dessin d'un cercle de rayon <r> en <x,y> dans le canevas <can>"
    can.create_oval(x-r, y-r, x+r, y+r, fill =coul)

class Application(Tk):
    def __init__(self):
        Tk.__init__(self)        # constructeur de la classe parente
        self.can =Canvas(self, width =475, height =130, bg ="white")
        self.can.pack(side =TOP, padx =5, pady =5)
        Button(self, text ="Train", command =self.dessine).pack(side =LEFT)
```

```
        Button(self, text ="Hello", command =self.coucou).pack(side =LEFT)
        Button(self, text ="Ecl34", command =self.eclai34).pack(side =LEFT)

    def dessine(self):
        "instanciation de 4 wagons dans le canevas"
        self.w1 = Wagon(self.can, 10, 30)
        self.w2 = Wagon(self.can, 130, 30, 'dark green')
        self.w3 = Wagon(self.can, 250, 30, 'maroon')
        self.w4 = Wagon(self.can, 370, 30, 'purple')

    def coucou(self):
        "apparition de personnages dans certaines fenêtres"
        self.w1.perso(3)        # 1er wagon, 3e fenêtre
        self.w3.perso(1)        # 3e wagon, 1e fenêtre
        self.w3.perso(2)        # 3e wagon, 2e fenêtre
        self.w4.perso(1)        # 4e wagon, 1e fenêtre

    def eclai34(self):
        "allumage de l'éclairage dans les wagons 3 & 4"
        self.w3.allumer()
        self.w4.allumer()

class Wagon(object):
    def __init__(self, canev, x, y, coul ='navy'):
        "dessin d'un petit wagon en <x,y> dans le canevas <canev>"
        # mémorisation des paramètres dans des variables d'instance :
        self.canev, self.x, self.y = canev, x, y
        # rectangle de base : 95x60 pixels :
        canev.create_rectangle(x, y, x+95, y+60, fill =coul)
        # 3 fenêtres de 25x40 pixels, écartées de 5 pixels :
        self.fen =[]    # pour mémoriser les réf. des fenêtres
        for xf in range(x +5, x +90, 30):
            self.fen.append(canev.create_rectangle(xf, y+5,
                                    xf+25, y+40, fill ='black'))
        # 2 roues, de rayon égal à 12 pixels  :
        cercle(canev, x+18, y+73, 12, 'gray')
        cercle(canev, x+77, y+73, 12, 'gray')

    def perso(self, fen):
        "apparition d'un petit personnage à la fenêtre <fen>"
        # calcul des coordonnées du centre de chaque fenêtre :
        xf = self.x + fen*30 -12
        yf = self.y + 25
        cercle(self.canev, xf, yf, 10, "pink")      # visage
        cercle(self.canev, xf-5, yf-3, 2)           # œil gauche
        cercle(self.canev, xf+5, yf-3, 2)           # œil droit
        cercle(self.canev, xf, yf+5, 3)             # bouche

    def allumer(self):
        "déclencher l'éclairage interne du wagon"
        for f in self.fen:
            self.canev.itemconfigure(f, fill ='yellow')

app = Application()
app.mainloop()
```

التمرين 13.10 :

```
#  Widget dérivé de <Canvas>, spécialisé pour
#  dessiner des graphiques élongation/temps
```

```
from tkinter import *
from math import sin, pi

class OscilloGraphe(Canvas):
    "Canevas spécialisé, pour dessiner des courbes élongation/temps"
    def __init__(self, master=None, larg=200, haut=150):
        "Constructeur de la base du graphique : quadrillage et axes"
        Canvas.__init__(self)                        # appel au constructeur
        self.configure(width=larg, height=haut)      # de la classe parente
        self.larg, self.haut = larg, haut            # mémorisation
        # tracé d'une échelle horizontale avec 8 graduations :
        pas = (larg-25)/8.               # intervalles de l'échelle horizontale
        for t in range(0, 9):
            stx = 10 + t*pas             # +10 pour partir de l'origine
            self.create_line(stx, haut/10, stx, haut*9/10, fill='grey')
        # tracé d'une échelle verticale avec 5 graduations :
        pas = haut*2/25.                 # intervalles de l'échelle verticale
        for t in range(-5, 6):
            sty = haut/2 -t*pas          # haut/2 pour partir de l'origine
            self.create_line(10, sty, larg-15, sty, fill='grey')
        self.traceAxes()                 # tracé des axes de référence X et Y

    def traceAxes(self):
        "Méthode traçant les axes de référence (pourra être surchargée)."
        # axes horizontal (X) et vertical (Y) :
        self.create_line(10, self.haut/2, self.larg, self.haut/2, arrow=LAST)
        self.create_line(10, self.haut-5, 10, 5, arrow=LAST)
        # indication des grandeurs physiques aux extrémités des axes :
        self.create_text(20, 10, anchor =CENTER, text = "e")
        self.create_text(self.larg-10, self.haut/2-12, anchor=CENTER, text="t")

    def traceCourbe(self, freq=1, phase=0, ampl=10, coul='red'):
        "tracé d'un graphique élongation/temps sur 1 seconde"
        curve =[]                       # liste des coordonnées
        pas = (self.larg-25)/1000.      # l'échelle X correspond à 1 seconde
        for t in range(0,1001,5):       # que l'on divise en 1000 ms.
            e = ampl*sin(2*pi*freq*t/1000 - phase)
            x = 10 + t*pas
            y = self.haut/2 - e*self.haut/25
            curve.append((x,y))
        n = self.create_line(curve, fill=coul, smooth=1)
        return n                        # n = numéro d'ordre du tracé

#### Code pour tester la classe : ####
if __name__ == '__main__':
    racine = Tk()
    gra = OscilloGraphe(racine, 250, 180)
    gra.pack()
    gra.configure(bg ='ivory', bd =2, relief=SUNKEN)
    gra.traceCourbe(2, 1.2, 10, 'purple')
    racine.mainloop()
```

التمرين 13.16 :

```
#  Tracé de graphiques élongation/temps pour 3
#  mouvements vibratoires harmoniques

from tkinter import *
from math import sin, pi
```

```
from exercice_13_10 import OscilloGraphe

class OscilloGrapheBis(OscilloGraphe):
    """Classe dérivée du widget Oscillographe (cf. exercice 13.10)"""
    def __init__(self, master =None, larg =200, haut =150):
        # Appel du constructeur de la classe parente :
        OscilloGraphe.__init__(self, master, larg, haut)

    def traceAxes(self):
        "Surchage de la méthode de même nom dans la classe parente"
        # tracé de l'axe de référence Y :
        pas = (self.larg-25)/8.       # intervalles de l'échelle horizontale
        self.create_line(10+4*pas, self.haut-5, 10+4*pas, 5, fill ='grey90',
                         arrow=LAST)
        # tracé de l'axe de référence X :
        self.create_line(10, self.haut/2, self.larg, self.haut/2,
                         fill= 'grey90', arrow=LAST)
        # indication des grandeurs physiques aux extrémités des axes :
        self.create_text(20+4*pas, 15, anchor=CENTER, text="e", fill='red')
        self.create_text(self.larg-5, self.haut/2-12, anchor=CENTER, text ="t",
                         fill='red')

class ChoixVibra(Frame):
    """Curseurs pour choisir fréquence, phase & amplitude d'une vibration"""
    def __init__(self, master=None, coul='red'):
        Frame.__init__(self)       # constructeur de la classe parente
        # Définition de quelques attributs d'instance :
        self.freq, self.phase, self.ampl, self.coul = 0, 0, 0, coul
        # Variable d'état de la case à cocher :
        self.chk = IntVar()                 # 'objet-variable' Tkinter
        Checkbutton(self, text='Afficher', variable=self.chk,
                    fg = self.coul, command=self.setCurve).pack(side=LEFT)
        # Définition des 3 widgets curseurs :
        Scale(self, length=150, orient=HORIZONTAL, sliderlength =25,
              label ='Fréquence (Hz) :', from_=1., to=9., tickinterval =2,
              resolution =0.25, showvalue =0,
              command = self.setFrequency).pack(side=LEFT, pady =5)
        Scale(self, length=150, orient=HORIZONTAL, sliderlength =15,
              label ='Phase (degrés) :', from_=-180, to=180, tickinterval =90,
              showvalue =0, command = self.setPhase).pack(side=LEFT, pady =5)
        Scale(self, length=150, orient=HORIZONTAL, sliderlength =25,
              label ='Amplitude :', from_=2, to=10, tickinterval =2,
              showvalue =0,
              command = self.setAmplitude).pack(side=LEFT, pady =5)

    def setCurve(self):
        self.master.event_generate('<Control-Z>')

    def setFrequency(self, f):
        self.freq = float(f)
        self.master.event_generate('<Control-Z>')

    def setPhase(self, p):
        pp =float(p)
        self.phase = pp*2*pi/360        # conversion degrés -> radians
        self.master.event_generate('<Control-Z>')

    def setAmplitude(self, a):
        self.ampl = float(a)
        self.master.event_generate('<Control-Z>')
```

```
## Classe principale ##

class ShowVibra(Frame):
    """Démonstration de mouvements vibratoires harmoniques"""
    def __init__(self, master=None):
        Frame.__init__(self)          # constructeur de la classe parente
        self.couleur = ['green', 'yellow', 'orange']
        self.trace = [0]*3            # liste des tracés (courbes à dessiner)
        self.controle = [0]*3         # liste des panneaux de contrôle
        # Instanciation du canevas avec axes X et Y :
        self.gra = OscilloGrapheBis(self, larg =400, haut=300)
        self.gra.configure(bg ='grey40', bd=3, relief=SUNKEN)
        self.gra.pack(side =TOP, pady=3)
        # Instanciation de 3 panneaux de contrôle (curseurs) :
        for i in range(3):
            self.controle[i] = ChoixVibra(self, self.couleur[i])
            self.controle[i].configure(bd =3, relief = GROOVE)
            self.controle[i].pack(padx =10, pady =3)
        # Désignation de l'événement qui déclenche l'affichage des tracés :
        self.master.bind('<Control-Z>', self.montreCourbes)
        self.master.title('Mouvements vibratoires harmoniques')
        self.pack()

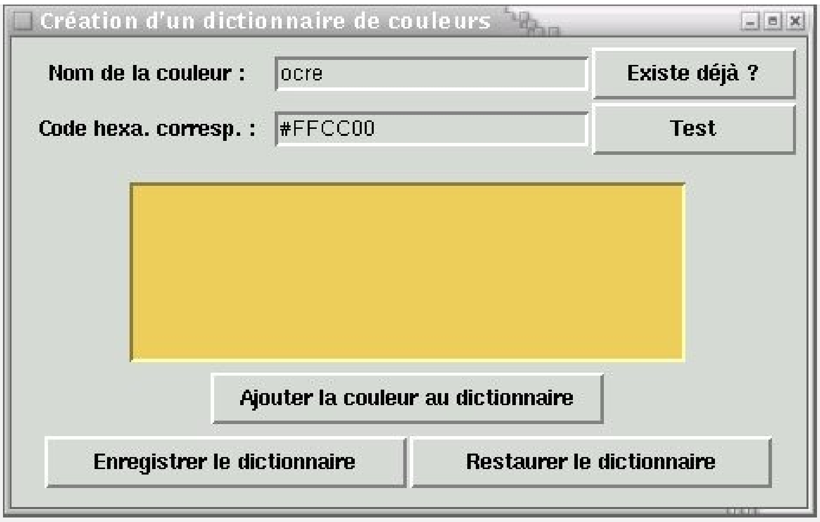
    def montreCourbes(self, event):
        """(Ré)Affichage des trois graphiques élongation/temps"""
        for i in range(3):
            # D'abord, effacer le tracé précédent (éventuel) :
            self.gra.delete(self.trace[i])
            # Ensuite, dessiner le nouveau tracé :
            if self.controle[i].chk.get():
                self.trace[i] = self.gra.traceCourbe(
                                    coul=self.couleur[i],
                                    freq=self.controle[i].freq,
                                    phase=self.controle[i].phase,
                                    ampl=self.controle[i].ampl)

#### Code de test : ###

if __name__ == '__main__':
    ShowVibra().mainloop()
```

التمرين 13.22 : قاموس ألوان

```
from tkinter import *
```




```
# Module donnant accès aux boîtes de dialogue standard pour
# la recherche de fichiers sur disque :
from tkinter.filedialog import asksaveasfile, askopenfile

class Application(Frame):
    '''Fenêtre d'application'''
    def __init__(self):
        Frame.__init__(self)
        self.master.title("Création d'un dictionnaire de couleurs")
        self.dico ={}            # création du dictionnaire

        # Les widgets sont regroupés dans deux cadres (Frames) :
        frSup =Frame(self)      # cadre supérieur contenant 6 widgets
        Label(frSup, text ="Nom de la couleur :",
              width =20).grid(row =1, column =1)
        self.enNom =Entry(frSup, width =25)         # champ d'entrée pour
        self.enNom.grid(row =1, column =2)          # le nom de la couleur
        Button(frSup, text ="Existe déjà ?", width =12,
               command =self.chercheCoul).grid(row =1, column =3)
        Label(frSup, text ="Code hexa. corresp. :",
              width =20).grid(row =2, column =1)
        self.enCode =Entry(frSup, width =25)        # champ d'entrée pour
        self.enCode.grid(row =2, column =2)         # le code hexa.
        Button(frSup, text ="Test", width =12,
               command =self.testeCoul).grid(row =2, column =3)
        frSup.pack(padx =5, pady =5)

        frInf =Frame(self)      # cadre inférieur contenant le reste
        self.test = Label(frInf, bg ="white", width =45,    # zone de test
                          height =7, relief = SUNKEN)
        self.test.pack(pady =5)
        Button(frInf, text ="Ajouter la couleur au dictionnaire",
               command =self.ajouteCoul).pack()
        Button(frInf, text ="Enregistrer le dictionnaire", width =25,
               command =self.enregistre).pack(side = LEFT, pady =5)
        Button(frInf, text ="Restaurer le dictionnaire", width =25,
               command =self.restaure).pack(side =RIGHT, pady =5)
        frInf.pack(padx =5, pady =5)
        self.pack()

    def ajouteCoul(self):
        "ajouter la couleur présente au dictionnaire"
```

```
        if self.testeCoul() ==0:        # une couleur a-t-elle été définie ?
            return
        nom = self.enNom.get()
        if len(nom) >1:                 # refuser les noms trop petits
            self.dico[nom] =self.cHexa
        else:
            self.test.config(text ="%s : nom incorrect" % nom, bg='white')

    def chercheCoul(self):
        "rechercher une couleur déjà inscrite au dictionnaire"
        nom = self.enNom.get()
        if nom in self.dico:
            self.test.config(bg =self.dico[nom], text ="")
        else:
            self.test.config(text ="%s : couleur inconnue" % nom, bg='white')

    def testeCoul(self):
        "vérifier la validité d'un code hexa. - afficher la couleur corresp."
        try:
            self.cHexa =self.enCode.get()
            self.test.config(bg =self.cHexa, text ="")
            return 1
        except:
            self.test.config(text ="Codage de couleur incorrect", bg ='white')
            return 0

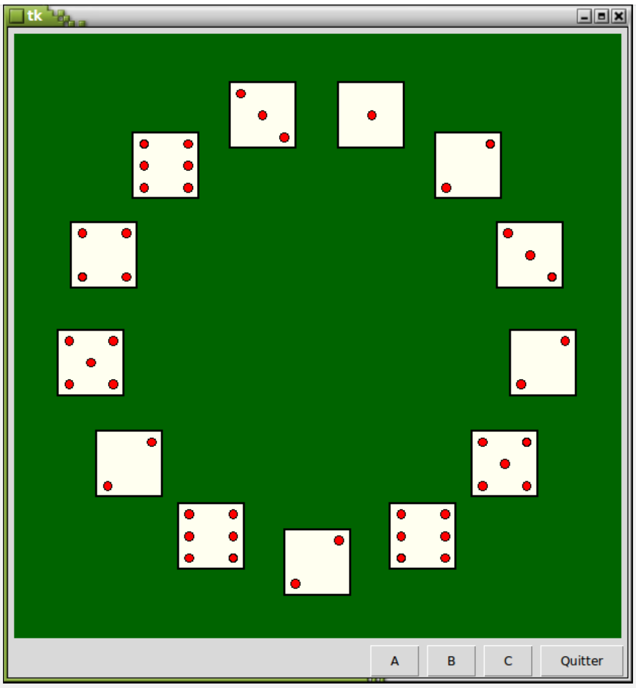
    def enregistre(self):
        "enregistrer le dictionnaire dans un fichier texte"
        # Cette méthode utilise une boîte de dialogue standard pour la
        # sélection d'un fichier sur disque. Tkinter fournit toute une série
        # de fonctions associées à ces boîtes, dans le module filedialog.
        # La fonction ci-dessous renvoie un objet-fichier ouvert en écriture :
        ofi =asksaveasfile(filetypes=[("Texte",".txt"),("Tous","*")])
        for clef, valeur in list(self.dico.items()):
            ofi.write("{0} {1}\n".format(clef, valeur))
        ofi.close()

    def restaure(self):
        "restaurer le dictionnaire à partir d'un fichier de mémorisation"
        # La fonction ci-dessous renvoie un objet-fichier ouvert en lecture :
        ofi =askopenfile(filetypes=[("Texte",".txt"),("Tous","*")])
        lignes = ofi.readlines()
        for li in lignes:
            cv = li.split()       # extraction de la clé et la valeur corresp.
            self.dico[cv[0]] = cv[1]
        ofi.close()

if __name__ == '__main__':
    Application().mainloop()
```

التمرين 13.23 (متنوع 3) :

```
from tkinter import *
from random import randrange
from math import sin, cos, pi
```




```
class FaceDom(object):
    def __init__(self, can, val, pos, taille =70):
        self.can =can
        x, y, c = pos[0], pos[1], taille/2
        self. carre = can.create_rectangle(x -c, y-c, x+c, y+c,
                                           fill ='ivory', width =2)
        d = taille/3
        # disposition des points sur la face, pour chacun des 6 cas :
        self.pDispo = [((0,0),),
                       ((-d,d),(d,-d)),
                       ((-d,-d), (0,0), (d,d)),
                       ((-d,-d),(-d,d),(d,-d),(d,d)),
                       ((-d,-d),(-d,d),(d,-d),(d,d),(0,0)),
                       ((-d,-d),(-d,d),(d,-d),(d,d),(d,0),(-d,0))]

        self.x, self.y, self.dim = x, y, taille/15
        self.pList =[]       # liste contenant les points de cette face
        self.tracer_points(val)

    def tracer_points(self, val):
        # créer les dessins de points correspondant à la valeur val :
        disp = self.pDispo[val -1]
        for p in disp:
            self.cercle(self.x +p[0], self.y +p[1], self.dim, 'red')
        self.val = val

    def cercle(self, x, y, r, coul):
        self.pList.append(self.can.create_oval(x-r, y-r, x+r, y+r, fill=coul))

    def effacer(self, flag =0):
        for p in self.pList:
            self.can.delete(p)
        if flag:
            self.can.delete(self.carre)

class Projet(Frame):
```

```
    def __init__(self, larg, haut):
        Frame.__init__(self)
        self.larg, self.haut = larg, haut
        self.can = Canvas(self, bg='dark green', width =larg, height =haut)
        self.can.pack(padx =5, pady =5)
        # liste des boutons à installer, avec leur gestionnaire :
        bList = [("A", self.boutA), ("B", self.boutB),
                 ("C", self.boutC), ("Quitter", self.boutQuit)]
        bList.reverse()         # inverser l'ordre de la liste
        for b in bList:
            Button(self, text =b[0], command =b[1]).pack(side =RIGHT, padx=3)
        self.pack()
        self.des =[]            # liste qui contiendra les faces de dés
        self.actu =0            # réf. du dé actuellement sélectionné

    def boutA(self):
        if len(self.des):
            return              # car les dessins existent déjà !
        a, da = 0, 2*pi/13
        for i in range(13):
            cx, cy = self.larg/2, self.haut/2
            x = cx + cx*0.75*sin(a)             # pour disposer en cercle,
            y = cy + cy*0.75*cos(a)             # on utilise la trigono !
            self.des.append(FaceDom(self.can, randrange(1,7) , (x,y), 65))
            a += da

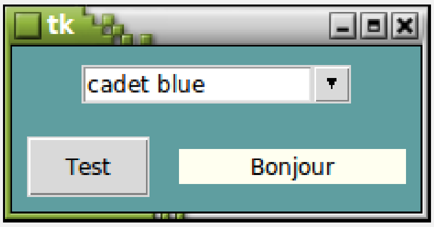
    def boutB(self):
        # incrémenter la valeur du dé sélectionné. Passer au suivant :
        v = self.des[self.actu].val
        v = v % 6
        v += 1
        self.des[self.actu].effacer()
        self.des[self.actu].tracer_points(v)
        self.actu += 1
        self.actu = self.actu % 13

    def boutC(self):
        for i in range(len(self.des)):
            self.des[i].effacer(1)
        self.des =[]
        self.actu =0

    def boutQuit(self):
        self.master.destroy()

Projet(600, 600).mainloop()
```

التمرين 14.1 (ودجة كومبوبوكس كاملة) :




```
    class ComboFull(Frame):
```

```
    "Widget composite 'Combo box' (champ d'entrée + liste 'déroulante')"
    def __init__(self, boss, item='', items=[], command ='', width =10,
                 listSize =5):
        Frame.__init__(self, boss)  # constructeur de la classe parente
        self.boss =boss             # référence du widget 'maître'
        self.items =items           # items à placer dans la boîte de liste
        self.command =command       # fonction à invoquer après clic ou <enter>
        self.item =item             # item entré ou sélectionné
        self.listSize =listSize     # nombre d'items visibles dans la liste
        self.width =width           # largeur du champ d'entrée (en caract.)

        # Champ d'entrée :
        self.entree =Entry(self, width =width)         # largeur en caractères
        self.entree.insert(END, item)
        self.entree.bind("<Return>", self.sortieE)
        self.entree.pack(side =LEFT)

        # Bouton pour faire apparaître la liste associée :
        self.gif1 = PhotoImage(file ="down.gif")       # ! variable persistante
        Button(self, image =self.gif1, width =15, height=15,
               command =self.popup).pack()

    def sortieL(self, event =None):
        # Extraire de la liste l'item qui a été sélectionné :
        index =self.bListe.curselection()       # renvoie un tuple d'index
        ind0 =int(index[0])                     # on ne garde que le premier
        self.item =self.items[ind0]
        # Actualiser le champ d'entrée avec l'item choisi :
        self.entree.delete(0, END)
        self.entree.insert(END, self.item)
        # Exécuter la commande indiquée, avec l'item choisi comme argument :
        self.command(self.item)
        self.pop.destroy()                      # supprimer la fenêtre satellite

    def sortieE(self, event =None):
        # Exécuter la commande indiquée, avec l'argument-item encodé tel quel :
        self.command(self.entree.get())

    def get(self):
        # Renvoyer le dernier item sélectionné dans la boîte de liste
        return self.item

    def popup(self):
        # Faire apparaître la petite fenêtre satellite contenant la liste.
        # On commence par récupérer les coordonnées du coin supérieur gauche
        # du présent widget dans la fenêtre principale :
        xW, yW =self.winfo_x(), self.winfo_y()
        # ... et les coordonnées de la fenêtre principale sur l'écran, grâce à
        # la méthode geometry() qui renvoie une chaîne avec taille et coordo. :
        geo =self.boss.geometry().split("+")
        xF, yF =int(geo[1]), int(geo[2])          # coord. coin supérieur gauche
        # On peut alors positionner une petite fenêtre, modale et sans bordure,
        # exactement sous le champ d'entrée :
        xP, yP = xF +xW +10, yF +yW +45     # +45 : compenser haut champ Entry
        self.pop =Toplevel(self)            # fenêtre secondaire ("pop up")
        self.pop.geometry("+{0}+{1}".format(xP, yP))    # positionnement / écran
        self.pop.overrideredirect(1)        # => fen. sans bordure ni bandeau
        self.pop.transient(self.master)     # => fen. 'modale'

        # Boîte de liste, munie d'un 'ascenseur' (scroll bar) :
        cadreLB =Frame(self.pop)            # cadre pour l'ensemble des 2
```

```
        self.bListe =Listbox(cadreLB, height=self.listSize, width=self.width-1)
        scrol =Scrollbar(cadreLB, command =self.bListe.yview)
        self.bListe.config(yscrollcommand =scrol.set)
        self.bListe.bind("<ButtonRelease-1>", self.sortieL)
        self.bListe.pack(side =LEFT)
        scrol.pack(expand =YES, fill =Y)
        cadreLB.pack()
        # Remplissage de la boîte de liste avec les items fournis :
        for it in self.items:
            self.bListe.insert(END, it)

if __name__ =="__main__":               # --- اختبار البرنامج ---
    def changeCoul(col):
        fen.configure(background = col)

    def changeLabel():
        lab.configure(text = combo.get())

    couleurs = ('navy', 'royal blue', 'steelblue1', 'cadet blue',
                'lawn green', 'forest green', 'yellow', 'dark red',
                'grey80','grey60', 'grey40', 'grey20', 'pink')
    fen =Tk()
    combo =ComboFull(fen, item ="néant", items =couleurs, command =changeCoul,
                    width =15, listSize =6)
    combo.grid(row =1, columnspan =2, padx =10, pady =10)
    bou = Button(fen, text ="Test", command =changeLabel)
    bou.grid(row =3, column =0, padx =8, pady =8)
    lab = Label(fen, text ="Bonjour", bg ="ivory", width =15)
    lab.grid(row =3, column =1, padx =8)
    fen.mainloop()
```

التمرين 16.1 (إنشاء قاعدة البيانات "الموسيقى") :

```
# Création et Alimentation d'une petite base de données SQLite

import sqlite3

# Établissement de la connexion - Création du curseur :
connex = sqlite3.connect("musique.sq3")
cur = connex.cursor()
# Création des tables. L'utilisation de try/except permet de réutiliser le
# script indéfiniment, même si la base de données existe déjà.
try:
    req ="CREATE TABLE compositeurs(comp TEXT, a_naiss INTEGER, "\
         "a_mort INTEGER)"
    cur.execute(req)
    req ="CREATE TABLE oeuvres(comp TEXT, titre TEXT, duree INTEGER, "\
         "interpr TEXT)"
    cur.execute(req)
except:
    pass                # Les tables existent certainement déjà => on continue.

print("Entrée des enregistrements, table des compositeurs :")
while 1:
    nom = input("Nom du compositeur (<Enter> pour terminer) : ")
    if nom =='':
        break
    aNais = input("Année de naissance : ")
    aMort = input("Année de mort : ")
    req ="INSERT INTO compositeurs (comp, a_naiss, a_mort) VALUES (?, ?, ?)"
```

```
    cur.execute(req, (nom, aNais, aMort))

print("Rappel des infos introduites :")
cur.execute("select * from compositeurs")
for enreg in cur:
    print(enreg)

print("Entrée des enregistrements, table des oeuvres musicales :")
while 1:
    nom = input("Nom du compositeur (<Enter> pour terminer) : ")
    if nom =='':
        break
    titre = input("Titre de l'oeuvre : ")
    duree = input("durée (minutes) : ")
    inter = input("interprète principal : ")
   req ="INSERT INTO oeuvres (comp, titre, duree, interpr) "\
         "VALUES (?, ?, ?, ?)"
    cur.execute(req, (nom, titre, duree, inter))

print("Rappel des infos introduites :")
cur.execute("select * from oeuvres")
for enreg in cur:
    print(enreg)

# Transfert effectif des enregistrements dans la BD :
connex.commit()
```

التمرين 18.3 :

```
# === Génération d'un document PDF avec gestion de fluables (paragraphes) ===

# Adaptations du script pour le rendre exécutable sous Python 2.6 ou 2.7 :
# (Ces lignes peuvent être supprimées si Reportlab est disponible pour Python3)
from __future__ import unicode_literals
from __future__ import division                # division "réelle"
from codecs import open                        # décodage des fichiers texte
# ---------------------------------------------------------------------------

# Importer quelques éléments de la bibliothèque ReportLab :
from reportlab.pdfgen.canvas import Canvas
from reportlab.lib.units import cm
from reportlab.lib.pagesizes import A4
from reportlab.platypus import Paragraph, Frame, Spacer
from reportlab.platypus.flowables import Image as rlImage
from reportlab.lib.styles import getSampleStyleSheet
from reportlab.lib.enums import TA_LEFT,TA_RIGHT,TA_JUSTIFY,TA_CENTER
from copy import deepcopy

styles = getSampleStyleSheet()              # dictionnaire de styles prédéfinis
styleN =styles["Normal"]                    # objet de classe ParagraphStyle()
styleM =deepcopy(styleN)                    # "vraie copie" d'un style
# Modification d'un de ces styles, pour disposer de deux variantes N et M :
styleN.fontName ='Helvetica-oblique'
styleN.fontSize =10
styleN.leading =11                          # interligne
styleN.alignment =TA_JUSTIFY                # ou TA_LEFT, TA_CENTER, TA_RIGHT
styleN.firstLineIndent =20                  # indentation de première ligne
styleN.textColor ='navy'

# Données à traiter :
```

```
fichier ="document_5.pdf"
bitmap ="bateau3.jpg"
dimX, dimY = 10*cm, 10*cm                    # dimensions imposées à l'image

# Construction de la liste de paragraphes <story> :
n, story = 1, []
ofi =open("document.txt", "r", encoding="Utf8")
while 1:
    ligne =ofi.readline()
    if not ligne:
        break
    # ajouter un paragraphe, dans un style différent une fois sur trois :
    if n %3 ==0:
        story.append(Paragraph(ligne, styleN))
    else:
        story.append(Paragraph(ligne, styleM))
    n +=1
ofi.close()

# === Construction du document PDF :
can = Canvas("%s" % (fichier), pagesize=A4)
largeurP, hauteurP = A4                 # largeur et hauteur de la page
can.setFont("Times-Bold", 18)
can.drawString(5*cm, 28*cm, "Gestion des paragraphes avec ReportLab")

# Mise en place de l'image, alignée à droite et centrée verticalement :
posX =largeurP -1*cm -dimX              # position du coin inférieur gauche
posY =(hauteurP -dimY)/2                # (on laisse une marge de 1 cm à droite)
can.drawImage(bitmap, posX, posY, width =dimX, height =dimY, mask="auto")

# Mise en place des trois cadres entourant l'image :
cS =Frame(1*cm, (hauteurP +dimY)/2, largeurP-2*cm, (hauteurP-dimY)/2-3*cm)
cM =Frame(1*cm, (hauteurP -dimY)/2, largeurP-2*cm-dimX, dimY)
cI =Frame(1*cm, 2*cm, largeurP-2*cm, (hauteurP-dimY)/2-2*cm)
# Mise en place des paragraphes (fluables) dans ces trois cadres :
cS.addFromList(story, can)                     # remplir le cadre supérieur
cM.addFromList(story, can)                     # remplir le cadre médian
cI.addFromList(story, can)                     # remplir le cadre inférieur
can.save()                                     # finaliser le document

print("Éléments restants dans <story> : {0}.".format(len(story)))
```

التمرين 18.4 :

```
############################################################################
# Modifications à apporter au script spectacles.py du chapitre 17 pour qu'il
# puisse produire une version imprimable (PDF) de la liste des réservations.
############################################################################

# Ajouter les importations suivantes pour l’utilisation sous Python 2 :
from __future__ import unicode_literals    # (celle-ci avant toutes les autres!)
from codecs import open

# Ajouter les importations suivantes pour pouvoir générer des documents PDF :
from reportlab.pdfgen.canvas import Canvas
from reportlab.lib.units import cm
from reportlab.lib.pagesizes import A4

# Dans la définition de la classe Glob(), modifier le nom du fichier annexe :
patronsHTML ="spectacles_2.htm"      # Fichier contenant les "patrons" HTML
```

```
# Ce fichier annexe sera une copie de "spectacles.htm", dans lequel on aura
# simplement modifié la rubrique suivante :

[*toutesReservations*]
<h4>Les réservations ci-après ont déjà été effectuées :</h4>
<p>{0}</p>
<h4><a href="{1}">Veuillez cliquer ici pour accéder au document PDF
correspondant.</A></h4>
##########

# Dans le corps de la méthode toutesReservations() de la classe WebSpectacles(),
# supprimer la dernière ligne "return mep(Glob.html[" ... etc",
# et la remplacer par le code ci-après :

        # ======= Construction du document PDF correspondant : =======
        # D'après le fichier de configuration tutoriel.conf, les documents
        # "statiques" doivent se trouver dans le sous-répertoire "annexes"
        # pour être accessibles depuis l'application web (mesure de sécurité) :
        fichier ="annexes/reservations.pdf"
        can = Canvas("%s" % (fichier), pagesize=A4)
        largeurP, hauteurP = A4                 # largeur et hauteur de la page
        # Dessin du logo (aligné par son coin inférieur gauche) :
        can.drawImage("annexes/python.gif", 1*cm, hauteurP-6*cm, mask="auto")
        can.setFont("Times-BoldItalic", 28)
        can.drawString(6*cm, hauteurP-6*cm, "Grand théâtre de Python city")
        # Tableau des réservations :
        posY =hauteurP-9*cm                     # position verticale de départ
        tabs =(1*cm, 7*cm, 11*cm, 16.5*cm)      # tabulations
        head =("Titre", "Nom du client", "Courriel", "Places réservées")
        # En-têtes du tableau :
        can.setFont("Times-Bold", 14)
        t =0
        for txt in head:
            can.drawString(tabs[t], posY, head[t])
            t +=1
        # Lignes du tableau :
        posY -=.5*cm
        can.setFont("Times-Roman", 14)
        for tupl in res:
            posY, t = posY-15, 0
            for champ in tupl:
                can.drawString(tabs[t], posY, str(champ))
                # (Les valeurs numériques doivent être converties en chaînes !)
                t +=1
        can.save()                                # Finalisation du PDF
        return mep(Glob.html["toutesReservations"].format(tabl, fichier))
```

التمرين 19.2 :

```
#####################################
# Bombardement d'une cible mobile   #
# (C) G. Swinnen - Avril 2004 - GPL #
#####################################

from tkinter import *
from math import sin, cos, pi
from random import randrange
from threading import Thread
import time                         # seulement pour le variante avec sleep()
```

```
class Canon:
    """Petit canon graphique"""
    def __init__(self, boss, num, x, y, sens):
        self.boss = boss              # référence du canevas
        self.num = num                # n° du canon dans la liste
        self.x1, self.y1 = x, y       # axe de rotation du canon
        self.sens = sens              # sens de tir (-1:gauche, +1:droite)
        self.lbu = 30                 # longueur de la buse
        # dessiner la buse du canon (horizontale) :
        self.x2, self.y2 = x + self.lbu * sens, y
        self.buse = boss.create_line(self.x1, self.y1,
                                      self.x2, self.y2, width =10)
        # dessiner le corps du canon (cercle de couleur) :
        self.rc = 15                  # rayon du cercle
        self.corps = boss.create_oval(x -self.rc, y -self.rc, x +self.rc,
                                       y +self.rc, fill ='black')
        # prédessiner un obus (au départ c'est un simple point) :
        self.obus = boss.create_oval(x, y, x, y, fill='red')
        self.anim = 0
        # retrouver la largeur et la hauteur du canevas :
        self.xMax = int(boss.cget('width'))
        self.yMax = int(boss.cget('height'))

    def orienter(self, angle):
        "régler la hausse du canon"
        # rem : le paramètre <angle> est reçu en tant que chaîne.
        # il faut donc le traduire en réel, puis le convertir en radians :
        self.angle = float(angle)*2*pi/360
        self.x2 = self.x1 + self.lbu * cos(self.angle) * self.sens
        self.y2 = self.y1 - self.lbu * sin(self.angle)
        self.boss.coords(self.buse, self.x1, self.y1, self.x2, self.y2)

    def feu(self):
        "déclencher le tir d'un obus"
        # référence de l'objet cible :
        self.cible = self.boss.master.cible
        if self.anim ==0:
            self.anim =1
            # position de départ de l'obus (c'est la bouche du canon) :
            self.xo, self.yo = self.x2, self.y2
            v = 20              # vitesse initiale
            # composantes verticale et horizontale de cette vitesse :
            self.vy = -v *sin(self.angle)
            self.vx = v *cos(self.angle) *self.sens
            self.animer_obus()

    def animer_obus(self):
        "animer l'obus (trajectoire balistique)"
        # positionner l'obus, en redéfinissant ses coordonnées :
        self.boss.coords(self.obus, self.xo -3, self.yo -3,
                                    self.xo +3, self.yo +3)
        if self.anim >0:
            # calculer la position suivante :
            self.xo += self.vx
            self.yo += self.vy
            self.vy += .5
            self.test_obstacle()        # a-t-on atteint un obstacle ?
            self.boss.after(15, self.animer_obus)
        else:
            # fin de l'animation :
```

```
            self.boss.coords(self.obus, self.x1, self.y1, self.x1, self.y1)

    def test_obstacle(self):
        "évaluer si l'obus a atteint une cible ou les limites du jeu"
        if self.yo >self.yMax or self.xo <0 or self.xo >self.xMax:
            self.anim =0
            return
        if self.yo > self.cible.y -3 and self.yo < self.cible.y +18 \
        and self.xo > self.cible.x -3 and self.xo < self.cible.x +43:
            # dessiner l'explosion de l'obus (cercle orange) :
            self.explo = self.boss.create_oval(self.xo -10,
                         self.yo -10, self.xo +10, self.yo +10,
                         fill ='orange', width =0)
            self.boss.after(150, self.fin_explosion)
            self.anim =0

    def fin_explosion(self):
        "effacer le cercle d'explosion - gérer le score"
        self.boss.delete(self.explo)
        # signaler le succès à la fenêtre maîtresse :
        self.boss.master.goal()

class Pupitre(Frame):
    """Pupitre de pointage associé à un canon"""
    def __init__(self, boss, canon):
        Frame.__init__(self, bd =3, relief =GROOVE)
        self.score =0
        s =Scale(self, from_ =88, to =65,
                 troughcolor ='dark grey',
                 command =canon.orienter)
        s.set(45)                          # angle initial de tir
        s.pack(side =LEFT)
        Label(self, text ='Hausse').pack(side =TOP, anchor =W, pady =5)
        Button(self, text ='Feu !', command =canon.feu).\
                             pack(side =BOTTOM, padx =5, pady =5)
        Label(self, text ="points").pack()
        self.points =Label(self, text=' 0 ', bg ='white')
        self.points.pack()
        # positionner à gauche ou à droite suivant le sens du canon :
        gd =(LEFT, RIGHT)[canon.sens == -1]
        self.pack(padx =3, pady =5, side =gd)

    def attribuerPoint(self, p):
        "incrémenter ou décrémenter le score"
        self.score += p
        self.points.config(text = ' %s ' % self.score)

class Cible:
    """objet graphique servant de cible"""
    def __init__(self, can, x, y):
        self.can = can             # référence du canevas
        self.x, self.y = x, y
        self.cible = can.create_oval(x, y, x+40, y+15, fill ='purple')

    def deplacer(self, dx, dy):
        "effectuer avec la cible un déplacement dx,dy"
        self.can.move(self.cible, dx, dy)
        self.x += dx
        self.y += dy
        return self.x, self.y
```

```
class Thread_cible(Thread):
    """objet thread gérant l'animation de la cible"""
    def __init__(self, app, cible):
        Thread.__init__(self)
        self.cible = cible          # objet à déplacer
        self.app = app              # réf. de la fenêtre d'application
        self.sx, self.sy = 6, 3     # incréments d'espace et de
        self.dt =300                # temps pour l'animation (ms)

    def run(self):
        "animation, tant que la fenêtre d'application existe"
        x, y = self.cible.deplacer(self.sx, self.sy)
        if x > self.app.xm -50 or x < self.app.xm /5:
                self.sx = -self.sx
        if y < self.app.ym /2 or y > self.app.ym -20:
                self.sy = -self.sy
        if self.app != None:
            self.app.after(int(self.dt), self.run)

    def stop(self):
        "fermer le thread si la fenêtre d'application est refermée"
        self.app =None

    def accelere(self):
        "accélérer le mouvement"
        self.dt /= 1.5
        self.app.bell()             # beep sonore

class Application(Frame):
    def __init__(self):
        Frame.__init__(self)
        self.master.title('<<< Tir sur cible mobile >>>')
        self.pack()
        self.xm, self.ym = 600, 500
        self.jeu = Canvas(self, width =self.xm, height =self.ym,
                          bg ='ivory', bd =3, relief =SUNKEN)
        self.jeu.pack(padx =4, pady =4, side =TOP)

        # Instanciation d'un canon et d'un pupitre de pointage :
        x, y = 30, self.ym -20
        self.gun =Canon(self.jeu, 1, x, y, 1)
        self.pup =Pupitre(self, self.gun)

        # instanciation de la cible mobile :
        self.cible = Cible(self.jeu, self.xm/2, self.ym -25)
        # animation de la cible mobile, sur son propre thread :
        self.tc = Thread_cible(self, self.cible)
        self.tc.start()
        # arrêter tous les threads lorsque l'on ferme la fenêtre :
        self.bind('<Destroy>',self.fermer_threads)

    def goal(self):
        "la cible a été touchée"
        self.pup.attribuerPoint(1)
        self.tc.accelere()

    def fermer_threads(self, evt):
        "arrêter le thread d'animation de la cible"
        self.tc.stop()
```

```
if __name__ =='__main__':
    Application().mainloop()
```

: ()Variante, utilisant une temporisation de la cible à l'aide de Time.sleep

```
class Thread_cible(Thread):
    """objet thread gérant l'animation de la cible"""
    def __init__(self, app, cible):
        Thread.__init__(self)
        self.cible = cible          # objet à déplacer
        self.app = app              # réf. de la fenêtre d'application
        self.sx, self.sy = 6, 3     # incréments d'espace et de
----->  self.dt =.3                 # temps pour l'animation

    def run(self):
        "animation, tant que la fenêtre d'application existe"
----->  while self.app != None:
            x, y = self.cible.deplacer(self.sx, self.sy)
            if x > self.app.xm -50 or x < self.app.xm /5:
                self.sx = -self.sx
            if y < self.app.ym /2 or y > self.app.ym -20:
                self.sy = -self.sy
--------->  time.sleep(self.dt)
```

